Progressive Uniform Manifold Approximation and Projection

Hyung-Kwon Ko, Jaemin Jo, and Jinwook Seo[†]

Seoul National University

Abstract

We present a progressive algorithm for the Uniform Manifold Approximation and Projection (UMAP), called the Progressive UMAP. Based on the theory of Riemannian geometry and algebraic topology, UMAP is an emerging dimensionality reduction technique that offers better versatility and stability than t-SNE. Although UMAP is also more efficient than t-SNE, it still suffers from an initial delay of a few minutes to produce the first projection, which limits its use in interactive data exploration. To tackle this problem, we improve the sequential computations in UMAP by making them progressive, which allows people to incrementally append a batch of data points into the projection at the desired pace. In our experiment with the Fashion MNIST dataset, we found that Progressive UMAP could generate the first approximate projection within a few seconds while also sufficiently capturing the important structures of the high-dimensional dataset.

CCS Concepts

• ***Human-centered computing*** → *Visual analytics;*

1. Introduction

We bring the Uniform Manifold Approximation and Projection (UMAP) [MHM18], a popular nonlinear dimensionality reduction technique, into Progressive Visual Analytics (PVA). For more than a decade, t-Distributed Stochastic Neighbor Embedding (t-SNE) [MH08] has been one of the most widely-used dimensionality reduction techniques. However, UMAP, which is based on Riemannian geometry and algebraic topology, has recently emerged as an alternative to t-SNE, offering better efficiency and applicability [EMK*19]. UMAP stands out for its fast computation time. It is approximately three times faster than the state-of-the-art t-SNE implementation [LRH*17] and has a better stability between runs and support for non-metric distance measures, such as the cosine distance and correlation distance. Since its introduction, UMAP has been quickly adopted by diverse disciplines, such as life sciences [BMH*19], physics [MBW*19], and computer science [HPRC19], attesting to its usefulness. Although performance benchmarks [MHM18] demonstrated that UMAP is much faster than t-SNE, it still is too slow to be used in interactive analysis effectively; when tested on a Macbook Pro with a 3.1 GHz Intel Core i7 and 8GB of RAM, UMAP took about 87 seconds to project the MNIST dataset onto a 2D space, far exceeding the time window of 10 seconds needed to keep the user's attention [Mil68, Shn84]. The problem compounds as users often have to run the algorithm several times to tune its hyperparameters. To address this issue, we present a progressive algorithm for UMAP (Progressive UMAP). After briefly introducing UMAP in Section 3, we identify a number of sequential computations in UMAP and explain how we improve each one by making it (Section 4). Finally, we show that Progressive UMAP can yield partial projections of data every few seconds with a quality comparable to the original UMAP (Section 5).

2. Related Work

A series of studies have introduced and refined the concept of PVA [SPG14, MPG*14, FP16, FFNS19] to manage the computational delay caused by a large amount of data or the high complexity of algorithms. Progressive computation is defined as a computation that reports intermediate outputs within a bounded latency that converges towards the true result with an ability to control the execution.

One of the popular applications of PVA is the progressive projection of high-dimensional data. Among the existing dimensionality reduction techniques, t-SNE [MH08] has received the most attention from PVA researchers. Kim et al. [KCL*17] introduced a per-iteration visualization environment in which users can interact in real time with algorithms that require complex computation, such as multidimensional scaling, t-SNE and latent Dirichlet allocation. Inspired by Ingram and Munzner's Q-SNE [IM15], Pezzotti et al. [PLvdM*16] presented a controllable t-SNE approximation, which enabled the interactive manipulation of t-SNE results, such as adding, removing and modifying data points. Similarly, Jo et al. [JSF18] proposed a progressive algorithm for indexing and querying the approximated k-nearest neighbors. They also introduced responsive t-SNE, an application of their algorithm, which

[†] Correspondence: jseo@snu.ac.kr

further reduced the initial delay of t-SNE using progressive neighbor computation. Motivated by these studies, we aim to develop a progressive algorithm for a popular dimensionality reduction technique, UMAP [MHM18], which is more efficient and flexible than t-SNE [EMK*19].

3. UMAP

Although UMAP [MHM18] is grounded in a complex mathematical foundation, its computation can be divided into two parts, graph construction and layout optimization, a configuration similar to t-SNE. In this section, we briefly explain the computation in an abstract manner. For more details about UMAP, please consult the original paper [MHM18].

Graph Construction: UMAP starts by generating a weighted k-nearest neighbor graph that describes the distances between data points in the high-dimensional space. Given an input dataset $X = \{x_1, \ldots, x_N\}$, the number of neighbors to consider k and a distance metric $d : X \times X \rightarrow [0, \infty)$, UMAP first computes $\mathcal{N}_i$, the k-nearest neighbors of x_i with respect to d. Then, UMAP computes two parameters, ρ_i and σ_i, for each data point x_i to identify its local metric space. ρ_i is a nonzero distance from x_i to its nearest neighbor:

$$\rho_i = \min_{j \in \mathcal{N}_i} \{d(x_i, x_j) \mid d(x_i, x_j) > 0\}, \tag{1}$$

and the σ_i that satisfies the condition below is found using binary search:

$$\sum_{j \in \mathcal{N}_i} \exp\left(\frac{-\max(0, d(x_i, x_j) - \rho_i)}{\sigma_i}\right) = \log_2(k). \tag{2}$$

Using ρ_i and σ_i, UMAP computes $v_{j|i}$, the weight of the edge from a point x_i to another point x_j:

$$v_{j|i} = \exp\left(\frac{-\max(0, d(x_i, x_j) - \rho_i)}{\sigma_i}\right). \tag{3}$$

To make it symmetric, UMAP computes a single edge with combined weight using $v_{j|i}$ and $v_{i|j}$:

$$v_{ij} = v_{j|i} + v_{i|j} - v_{j|i} \cdot v_{i|j}. \tag{4}$$

Note that v_{ij} indicates the similarity between points x_i and x_j in the original space. Let y_i be the projection of x_i in a low-dimensional projection space. The similarity between two projected points y_i and y_j is:

$$w_{ij} = (1 + a||y_i - y_j||_2^{2b})^{-1}, \tag{5}$$

where a and b are positive constants defined by the user. Setting both a and b to 1 is identical to using Student's t-distribution to measure the similarity between two points in the projection space as in t-SNE [MH08].

Layout Optimization: The goal of layout optimization is to find the y_i that minimizes the difference (or loss) between v_{ij} and w_{ij}. In contrast to t-SNE where the Kullback-Leibler divergence between v_{ij} and w_{ij} is measured as the loss of the projection, UMAP measures the cross entropy between v_{ij} and w_{ij}:

Algorithm 1: Progressive UMAP

```
Procedure ProgressiveUMAP ($X$, $num\_iterations$, $ops$)
    KNNTable ← new KNNTable(X), iterations ← 0;
    while $iterations{+}{+} < num\_iterations$ do
        if $size(KNNTable) < size(X)$ then
            $X_{new}, X_{updated}$ = KNNTable.run($ops$);
            set initial $y_i$ for points in $X_{new}$;
            update $\rho_i$ (Equation 1) and $\sigma_i$ (Equation 2);
            compute $v_{j|i}$ (Equation 3);
            compute $v_{ij}$ (Equation 4);
        compute $\frac{C_{UMAP}}{y_i}$ (Equation 7, Equation 8);
        update $y_i$;
    end
    return $y_i$;
```

$$C_{UMAP} = \sum_{i \neq j} [v_{ij} \cdot \log(\frac{v_{ij}}{w_{ij}}) - (1 - v_{ij}) \cdot \log(\frac{1 - v_{ij}}{1 - w_{ij}})]. \tag{6}$$

The authors of UMAP argued that UMAP returns clearer separation between clusters than t-SNE since C_{UMAP} gives penalties for forming both local and global structures.

y_i is initialized through spectral embedding [BN02] and iteratively optimized to minimize C_{UMAP}. Given the output weight w_{ij} as $1/(1 + ad_{ij}^{2b})$, the attractive gradient is:

$$\frac{{C_{UMAP}}^{+}}{y_i} = \frac{-2abd_{ij}^{2(b-1)}}{1 + ad_{ij}^{2b}} v_{ij}(y_i - y_j), \tag{7}$$

and repulsive gradient is:

$$\frac{{C_{UMAP}}^{-}}{y_i} = \frac{2b}{(\varepsilon + d_{ij}^2)(1 + ad_{ij}^{2b})} (1 - v_{ij})(y_i - y_j), \tag{8}$$

where ε is a small value added to prevent division by zero and d_{ij} is a Euclidean distance between y_i and y_j. For efficient optimization, UMAP employs the negative sampling technique from Word2Vec [MSC*13]. It chooses a target point and the negative samples of the point, where the former updates the position of the target with the attractive gradient and the latter do so with the repulsive gradient. Moreover, UMAP also employs edge sampling [TQW*15, TLZM16] to simplify and accelerate the optimization process, which will be explained in Section 5 in detail.

4. Progressive UMAP

We found that the current implementation of UMAP [MHM18] could suffer a long initial delay depending on the size of the dataset, because it only works on a fixed set of data points with no support for adding new points progressively. In this section, we elaborate on our novel algorithm, Progressive UMAP (Algorithm 1), which allows users to feed small batches of data points into UMAP incrementally to obtain the desired latency between intermediate projection outputs. To this end, we identify sequential procedures in

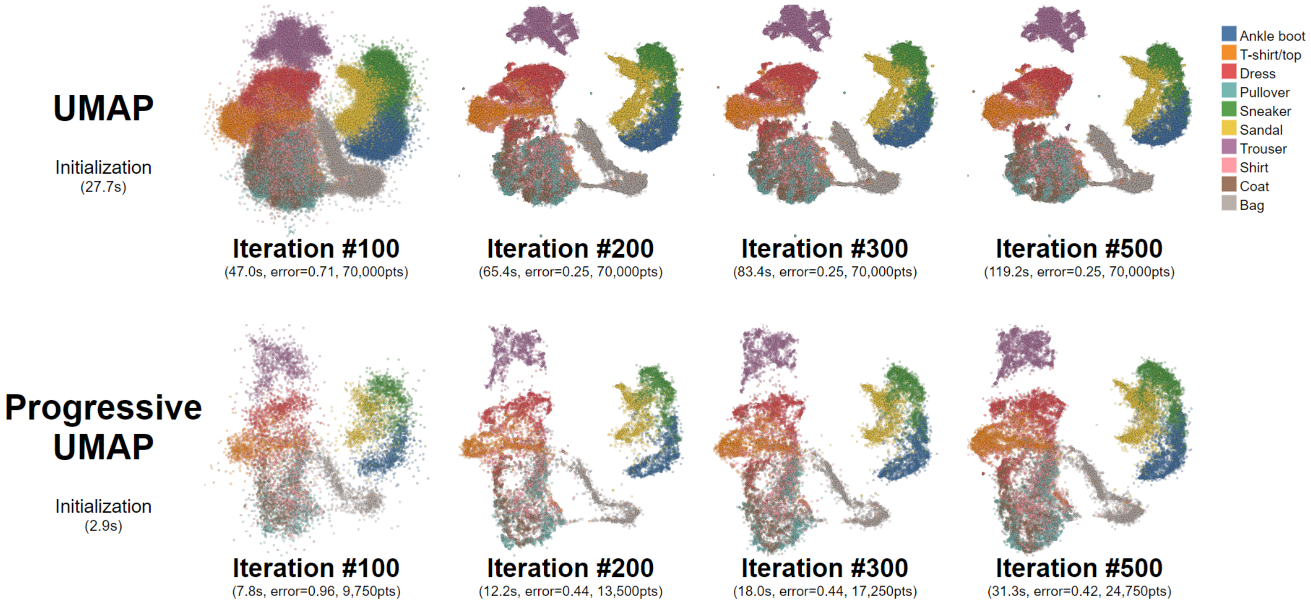

Figure 1: ***Projection of the Fashion MNIST dataset using UMAP and Progressive UMAP.*** *Seventy thousand data points (784 dimensions) of the Fashion MNIST dataset have been projected onto a 2D space. UMAP took 27.7 s for initialization (i.e., neighbor computation and graph construction) that can interrupt interactive data exploration. In contrast, Progressive UMAP allowed users to project a small fraction of the points with shorter initialization time (2.9 s) and progressively append the remaining points to the projection. Although UMAP produced a smaller average loss at the end of iterations, Progressive UMAP could project the data points in a reasonable time-bound, sufficiently capturing the characteristics of the data.*

the original UMAP algorithm and transform them into progressive procedures.

Computing $\mathcal{N}_i$: To build and maintain the k-nearest neighbor graph, we leverage the KNN lookup table from PANENE [JSF18]. PANENE employs randomized k-d trees [ML09] to approximate and update the k-nearest neighbors of an increasing number of data points. PANENE accepts a parameter called *ops* that indicates the allowed number of tasks per iteration that can be controlled to find the balance between latency and accuracy. For example, setting *ops* to a larger number will index more points per iteration, which will yield a more accurate projection at the cost of longer latency. Progressive UMAP starts with calling the update procedure of the KNN lookup table which returns two sets of points: X_{new} for newly inserted points and $X_{updated}$ for points whose neighbors are changed due to the insertion.

Computing ρ_i and σ_i: For every data point in $X_{updated}$ and X_{new}, we recompute ρ_i and σ_i according to Equation 1 and Equation 2. For space efficiency, UMAP used the coordinate list (COO) that only stores row, column, and value information as a list of tuples. Progressive UMAP updates the COO recalculating $v_{j|i}$ (Equation 3) for the selected points – $X_{updated}$ and X_{new} – and changing the corresponding values if there is a change from the previous ones. Last, we make the COO symmetric (Equation 4).

Layout Initialization: Although spectral embedding produces an effective initial projection, its quadratic time-complexity causes severe delay. Progressive UMAP initializes the positions of newly inserted points in two stages. For the first batch of points, 1) we run the algorithm with a large value of *ops* (e.g., 15,000), using the same spectral embedding technique as UMAP. Since we start with a relatively small number of points, this would take much less time than the original UMAP's spectral embedding. Hereafter, 2) we lower the value of *ops* (e.g., 1,000) not to focus on the appending process but to obtain an optimized projection output fast. Starting with the second batch, we set the initial projected position of each newly inserted point equal to its closest neighbor's position disturbed by a small Gaussian random noise to prevent collisions.

Layout Optimization: Analogously, we go through two stages for layout optimization. As it affects the overall time for convergence and increases the stability of the final output, it is very important to position the first batch of points well so that clusters are unambiguously separated. To this end, 1) we run more iterations (e.g., 40) in the first batch so each cluster can settle its position. Afterwards, 2) we run fewer iterations (e.g., 4) to focus on attaining the projection result fast. However, users can control the number of iterations for each stage; for example, if the size of data is small enough, they can set the algorithm to use the same number of iterations for both stages.

Based on the original UMAP implementation [MHM], our Progressive UMAP is written in Python. We leveraged PyNENE, a Python binding of PANENE [JSF18], for the KNN lookup table as well as Numba [LPS15] for parallel computation of distances, graph weights and optimization. The source code is available at `https://www.github.com/hyungkwonko/progressive-umap`.

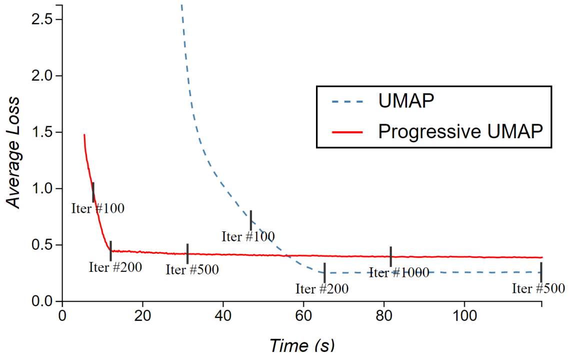

Figure 2: ***Average loss over time*** *UMAP generated its first projection with an initial delay of 27.7 s (dotted line) while Progressive UMAP minimized the delay to 2.9 s. The loss of Progressive UMAP converged to 0.42 which was higher than that of UMAP (0.25). For ease of comparison, the loss at 100, 200 and 500 iterations was marked with grey vertical lines. After all points are added, the final loss of Progressive UMAP was 0.38.*

5. Evaluation and Discussion

To measure the loss of our method and UMAP at run-time we employ the objective function suggested by Tang et al. [TQW*15]. In their study, the authors proposed an objective function that subsumes all the edges (both observed and unobserved) to optimize a graph layout. As summing up all the edges in a complete graph is computationally expensive, they further suggested a fast method based on edge sampling, described as follows. We randomly sample edges with a probability proportional to their weights. We subsequently treat the selected ones as binary edges. The objective function is:

$$O = \sum_{(i,j)\in E} v_{ij}(\log(w_{ij}) + \sum_{k=1}^{M} E_{j_k \sim P_n(j)} \gamma \log(1 - w_{ij_k})). \qquad (9)$$

Here, v_{ij} and w_{ij} are the similarities in the high and low-dimensional spaces respectively, M is the number of negative samples and $E_{j_k \sim P_n(j)}$ indicates that j_k is sampled according to a noisy distribution, $P_n(j)$, from Word2Vec [MSC*13]. However, applying the objective function directly to both algorithms induces bias since the total numbers of edges in UMAP and Progressive UMAP are different when data points are being inserted progressively in Progressive UMAP. To make the calculations comparable, we divide the loss by the number of sampled edges, a quantity which we will call unbiased loss or *average loss*.

For the evaluation, we ran both UMAP and Progressive UMAP on the Fashion MNIST dataset [XRV17] which has 70,000 rows of 784 dimensions (28 × 28), each row describing an item from 10 classes (e.g., t-shirt, trouser, etc.). As a baseline, we used the original UMAP implementation [MHM]. Both algorithms were tested on a machine equipped with an Intel Core i7-4790K CPU (4.0 GHz) and 16 GB of main memory.

UMAP has several important hyperparameters: k (the number of neighbors to consider), *min_dist* (the minimum distance between points in the low-dimensional space) and *metric* (a metric to compute the distance between points in the high-dimensional space). We set k to 5, *min_dist* to 0.1, and *metric* to the Euclidean distance.

Figure 1 shows intermediate 2D projection outputs generated by each algorithm over time. Each point denotes a single row of 784 dimensions in the high-dimensional space, color-coded by class. The initialization process of UMAP took 27.7 seconds as it considered all the data points at the beginning. In contrast, Progressive UMAP took 2.9 seconds to initialize because it could incrementally append data points in later iterations. Similarly, the time required to reach the same *average loss* was faster in Progressive UMAP; Progressive UMAP took 11.1 seconds to obtain the *average loss* of 0.5, while the original UMAP took 52.8 seconds (Figure 2). Although Progressive UMAP produced a bigger *average loss* at the end, it located points much faster than UMAP and converged in a reasonable time-bound, sufficiently capturing the important characteristics of the dataset; for example, the intermediate outputs of Progressive UMAP at the 200th iteration (Figure 1) manifest a clear separation of data points between clusters.

Next, we tested the effect of *ops* (i.e., the parameter passed to the KNN lookup table) and the number of iterations on optimizing the first projection. We gradually changed *ops* from 300 to 1,000 with an interval of 100. We found it is possible to control the computation time of an iteration by changing *ops*. However, if *ops* is too small, it can harm the stability of the projection result since, if we insert too few points at a time, it is impossible to choose the nearest neighbors that capture the local manifold robustly. We also examined the effect of the number of iterations on optimizing the projection (y_i) for the first batch by changing it from 20 to 60 with an interval of 10. As expected, keeping the number of iterations small in the first batch worsened the convergence speed and stability of the projection result. On the other hand, having a large number of iterations in the first batch helped achieve better projection quality but also slowed down convergence. The results of these experiments are also available in our repository.

To sum up, we found that Progressive UMAP is able to not only generate intermediate projection outputs whose quality is comparable to the original UMAP within a reasonable time-bound but also provides an ability to control the trade-off between computation time and the quality of projection.

6. Conclusion and Future Work

We present a progressive algorithm for the Uniform Manifold Approximation and Projection (Progressive UMAP). Through our quantitative evaluation, we found that Progressive UMAP can generate the approximate projection and update it every few seconds. But, in the current implementation, not all computation steps are bounded (e.g., the matrix multiplication) in time. For this reason, the algorithm might not be able to handle a dataset that is too large. In future work, we plan to identify such problematic computational steps and speed them up using, for example, hardware acceleration.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1A2C2089062).

References

[BMH*19] BECHT E., MCINNES L., HEALY J., DUTERTRE C.-A., KWOK I. W., NG L. G., GINHOUX F., NEWELL E. W.: Dimensionality reduction for visualizing single-cell data using umap. *Nature biotechnology 37*, 1 (2019), 38. 1

[BN02] BELKIN M., NIYOGI P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems* (2002), pp. 585–591. 2

[EMK*19] ESPADOTO M., MARTINS R. M., KERREN A., HIRATA N. S., TELEA A. C.: Towards a quantitative survey of dimension reduction techniques. *IEEE Transactions on Visualization and Computer Graphics* (2019). 1, 2

[FFNS19] FEKETE J.-D., FISHER D., NANDI A., SEDLMAIR M.: Progressive Data Analysis and Visualization (Dagstuhl Seminar 18411). *Dagstuhl Reports 8*, 10 (2019), 1–40. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10346, doi:10.4230/DagRep.8.10.1. 1

[FP16] FEKETE J.-D., PRIMET R.: Progressive analytics: A computation paradigm for exploratory data analysis. *arXiv preprint arXiv:1607.05162* (2016). 1

[HPRC19] HOHMAN F., PARK H., ROBINSON C., CHAU D. H. P.: S ummit: Scaling deep learning interpretability by visualizing activation and attribution summarizations. *IEEE transactions on visualization and computer graphics 26*, 1 (2019), 1096–1106. 1

[IM15] INGRAM S., MUNZNER T.: Dimensionality reduction for documents with nearest neighbor queries. *Neurocomputing 150* (2015), 557–569. 1

[JSF18] JO J., SEO J., FEKETE J.-D.: Panene: A progressive algorithm for indexing and querying approximate k-nearest neighbors. *IEEE transactions on visualization and computer graphics* (2018). 1, 3

[KCL*17] KIM H., CHOO J., LEE C., LEE H., REDDY C. K., PARK H.: Pive: Per-iteration visualization environment for real-time interactions with dimension reduction and clustering. In *Thirty-First AAAI Conference on Artificial Intelligence* (2017). 1

[LPS15] LAM S. K., PITROU A., SEIBERT S.: Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC* (2015), pp. 1–6. 3

[LRH*17] LINDERMAN G. C., RACHH M., HOSKINS J. G., STEINERBERGER S., KLUGER Y.: Efficient algorithms for t-distributed stochastic neighborhood embedding. *arXiv preprint arXiv:1712.09005* (2017). 1

[MBW*19] MEHTA P., BUKOV M., WANG C.-H., DAY A. G., RICHARDSON C., FISHER C. K., SCHWAB D. J.: A high-bias, low-variance introduction to machine learning for physicists. *Physics reports* (2019). 1

[MH08] MAATEN L. V. D., HINTON G.: Visualizing data using t-sne. *Journal of machine learning research 9*, Nov (2008), 2579–2605. 1, 2

[MHM] MCINNES L., HEALY J., MELVILLE J.: Umap. https://github.com/lmcinnes/umap, last accessed: 2020-02-20. 3, 4

[MHM18] MCINNES L., HEALY J., MELVILLE J.: Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018). 1, 2

[Mil68] MILLER R. B.: Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I* (1968), pp. 267–277. 1

[ML09] MUJA M., LOWE D. G.: Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1) 2*, 331-340 (2009), 2. 3

[MPG*14] MÜHLBACHER T., PIRINGER H., GRATZL S., SEDLMAIR M., STREIT M.: Opening the black box: Strategies for increased user involvement in existing algorithm implementations. *IEEE transactions on visualization and computer graphics 20*, 12 (2014), 1643–1652. 1

[MSC*13] MIKOLOV T., SUTSKEVER I., CHEN K., CORRADO G. S., DEAN J.: Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (2013), pp. 3111–3119. 2, 4

[PLvdM*16] PEZZOTTI N., LELIEVELDT B. P., VAN DER MAATEN L., HÖLLT T., EISEMANN E., VILANOVA A.: Approximated and user steerable tsne for progressive visual analytics. *IEEE transactions on visualization and computer graphics 23*, 7 (2016), 1739–1752. 1

[Shn84] SHNEIDERMAN B.: Response time and display rate in human performance with computers. *ACM Computing Surveys (CSUR) 16*, 3 (1984), 265–285. 1

[SPG14] STOLPER C. D., PERER A., GOTZ D.: Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (2014), 1653–1662. 1

[TLZM16] TANG J., LIU J., ZHANG M., MEI Q.: Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th international conference on world wide web* (2016), pp. 287–297. 2

[TQW*15] TANG J., QU M., WANG M., ZHANG M., YAN J., MEI Q.: Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web* (2015), pp. 1067–1077. 2, 4

[XRV17] XIAO H., RASUL K., VOLLGRAF R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. arXiv:cs.LG/1708.07747. 4