Big Data Research 26 (2021) 100254

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Scaling the Growing Neural Gas for Visual Cluster Analysis

Elio Ventocilla [a,*], Rafael M. Martins [b], Fernando Paulovich [c], Maria Riveiro [d]

[a] School of Informatics, University of Skövde, Sweden
[b] Department of Computer Science and Media Technology, Linnaeus University, Sweden
[c] Faculty of Computer Science, Dalhousie University, Canada
[d] School of Engineering, Jönköping University, Sweden

ARTICLE INFO

Article history:
Received 1 October 2020
Received in revised form 18 May 2021
Accepted 12 June 2021
Available online 12 August 2021

Keywords:
Growing neural gas
Big data
Visual analytics
Unsupervised learning
Exploratory data analysis

ABSTRACT

The growing neural gas (GNG) is an unsupervised topology learning algorithm that models a data space through interconnected units that stand on the most populated areas of that space. Its output is a graph that can be visually represented on a two-dimensional plane, disclosing cluster patterns in datasets. It is common, however, for GNG to result in highly connected graphs when trained on high-dimensional data, which in turn leads to highly cluttered 2D representations that may fail to disclose meaningful patterns. Moreover, its sequential learning limits its potential for faster executions on local datasets, and, more importantly, its potential for training on distributed datasets while leveraging from the computational resources of the infrastructures in which they reside.

This paper presents two methods that improve GNG for the visualization of cluster patterns in large-scale and high-dimensional datasets. The first one focuses on providing more accurate and meaningful 2D visual representations for cluster patterns of high-dimensional datasets, by avoiding connections that lead to high-dimensional graphs in the modeled topology which may, in turn, result in overplotting and clutter. The second method presented in this paper enables the use of GNG on big and distributed datasets with faster execution times, by modeling and merging separate parts of a dataset using the MapReduce model.

Quantitative and qualitative evaluations show that the first method leads to the creation of lower-dimensional graph structures that provide more meaningful (and sometimes more accurate) cluster representations with less overplotting and clutter; and that the second method preserves the accuracy and meaning of the cluster representations while enabling its execution in large-scale and distributed settings.

1. Introduction

A common problem in exploratory data analysis—a process that relies on limited preconceived assumptions [1]—is the investigation of cluster patterns in datasets [2], i.e., uncovering groups of instances which form neighborhoods according to a given similarity or distance metric. One way to uncover such patterns is by *modeling* those neighborhoods and then *visually encoding* them. Modeling is often achieved by using dimensionality reduction (DR) (e.g., PCA [3], t-SNE [4], and UMAP [5]) or clustering (e.g., Ward [6], OPTICS [7], and SOM [8]) techniques; whereas visual encoding can be achieved by employing scatter plots, dendrograms, reachability plots, and U-Matrices.

As datasets grow in terms of size (number of data points) and dimensionality (number of features), it becomes more challenging to uncover such cluster patterns in a human-interpretable and usable way. High-dimensional datasets are more difficult to model because data points are sparser in the multidimensional space (a.k.a., the curse of dimensionality [9]). This makes modeled distances less meaningful, therefore affecting the representativeness of the subsequent visual representations. Bigger datasets (i.e. with millions of data points) entail even more usability challenges, such as avoiding overplotting and clutter in the visual representations or providing system feedback from both learning processes and user-triggered interactions under a given time threshold (i.e., 10 seconds for task-preserving latency) [10–12].

The DR community has made extensive progress in modeling high dimensional spaces, as well as in enabling faster and progressive computations (see [13–15] for thorough surveys). Most techniques, however, have a limit on the number of data points that can be modeled before producing overlapping elements and clut-

* Corresponding author.
E-mail address: elio.ventocilla@his.se (E. Ventocilla).

https://doi.org/10.1016/j.bdr.2021.100254

(a) GNG (b) t-SNE

Fig. 1. Projections of the MNIST dataset [21] using GNG and t-SNE.

ter in the visual encoding, or before exceeding the recommended feedback times. This is mainly due to the nature of DR techniques, where all data points are either visually encoded or need to be accounted for during each training epoch.

Some clustering techniques, such as SOM and the GNG [16], resolve these issues by quantizing the data, i.e., by *compressing* the data into a set of representative units, and by working in an incremental manner, i.e., modeling the space by taking subsets of data at a time. The former reduces the occurrence of overplotting and visual clutter, as well as computational burdens triggered by user interactions such as selecting and zooming, by only depicting a smaller set of representative units. The latter, on the other hand, allows systems to provide faster feedback from ongoing learning processes, while also allowing them to model streaming data. Additionally, the output from such techniques lends itself to subsequent analytical tasks such as anomaly detection, e.g., [17].

Despite their similarities, some works have described advantages of GNG over SOM. First, the visual encoding of SOMs (U-Matrices [18]) was empirically shown to be less intuitive than GNG's graphs [19]. Second, GNG's models can partition data into independent networks which reflect the number of clusters in a dataset, hence providing an automated estimate of this hyper-parameter (usually known as k) [20]. Taking these benefits into account, this paper proposes two methods to further improve GNG's modeling power and efficiency when it comes to visually uncovering cluster patterns in large and high dimensional datasets.

1.1. Problem description

In spite of the previously described benefits, visual representations of GNG tend to degrade as datasets grow in dimensionality [19]. Concretely, GNG models of high-dimensional data result in highly connected networks which tend to produce many overlapping units and edges when projected on a two-dimensional plane, resembling hairball-like networks (see Fig. 1) with arguably no visible cluster patterns as compared to other techniques (e.g., t-SNE).

Moreover, GNG's sequential implementation can create a bottleneck for large and distributed datasets. In such settings, where data resides on different machines (nodes), a system might be required to retrieve the data from each of the nodes in order to train the model on a single machine. This could work well by sampling data points in an iterative manner (as suggested in [22]) but would arguably create an unnecessary burden on the system in terms of network traffic, thus potentially increasing feedback times beyond the recommended usability thresholds (especially when dealing with high-dimensional datasets such as images).

1.2. Contribution

This paper describes two sets of updates for the GNG algorithm, aimed at resolving the two aforementioned problems. Namely, avoiding *entanglement* in GNG's models to provide more accurate and meaningful cluster pattern representations of high-dimensional datasets; and parallelizing the learning process (using the MapReduce programming model [23]) so that it may produce faster results in either distributed or local datasets. To assess the validity of these solutions, three different GNG versions were created: **u-GNG** to assess entanglement prevention, **parGNG** to assess parallelization, and **u-parGNG** to assess both. In addition to these, one other version was assessed, **s-GNG**, which adds sampling to u-parGNG to further decrease execution times and improve its modeling by avoiding noise. The performance of these four versions, along with the original GNG as a baseline, were assessed quantitatively in terms of execution times, mean squared error (MSE), and three other metrics aimed at evaluating a model's cluster separation in terms of the connections among its units. Their cluster pattern representations were also qualitatively assessed by comparing their two-dimensional visual representations (embeddings).

Results show that (a) u-GNG creates more accurate and meaningful visual representations of cluster patterns within similar execution times; (b) parGNG scales to distributed datasets while preserving, and at times improving, GNG's modeling accuracy; (c) u-parGNG also preserves u-GNG's modeling accuracy while achieving faster results; and (d) s-GNG can further lower u-parGNG's learning times while preserving, and sometimes improving, its modeling power.

The rest of the paper is organized as follows: the next section provides a brief introduction to GNG and MapReduce, while related work is reviewed in Section 3. The two new methods are presented in Sections 4 and 5, while the experimental setup and results are described in Sections 6 and 7. These are followed by a discussion of the results and a brief concluding section.

2. Background

2.1. The growing neural gas

Fritzke [16] proposed the GNG as a learning algorithm that finds "a topological structure which closely reflects the topology of the data distribution". Unlike the Neural Gas (NG), proposed by Martinetz et al. [24], GNG uses Competitive Hebbian learning [25] to dynamically add units to the modeling network during the learning process—hence Growing NG. The outcome of a GNG is a set of units (neurons), each with an associated weight vector (prototype) of the same dimensionality as the input data, and a set of edges connecting the units.

The learning process is composed of the following steps:

1. Start with two connected units with randomly initialized prototypes.
2. Take an input signal (data point) δ from the dataset.
3. Find the two closest units μ_a and μ_b, based on the Euclidean distance of their prototypes to δ.
4. Linearly increment the age of all edges connected to μ_a.
5. Add (accumulate) the squared Euclidean distance ($||\mu_a - \delta||^2$) to an error variable associated to μ_a.
6. Move μ_a, and its topological neighbors, towards δ by a factor ϵ_b and ϵ_n respectively.
7. Connect μ_a and μ_b if they are not already connected; set the age value of their connecting edge to zero if they already are.
8. Remove all edges with an age over a threshold a_{max}, as well as all resulting unconnected units.
9. At every λ input signal, insert a new unit between the unit μ_q with the largest accumulated error, and its direct topological neighbor μ_r with the largest accumulated error. The new unit's prototype is set to stand in between, i.e., $0.5(\mu_q - \mu_q r)$. Reduce the errors of μ_q and μ_r by a factor α. Set the new unit's error equal to μ_q's error.
10. Decrease all units' error by d.
11. If no stopping criteria are met, go to step 2.

This process is incremental in nature, since learning is achieved one data point (signal) at a time. This allows GNG's inclusion in progressive and interactive systems, where analysts can steer and explore the modeled space, as training proceeds [12,26]. Concretely, a system can provide partial results at every given time step t, so that users may explore, assess, and steer the modeling process. Another inherent benefit is that it can model clusters of different shapes. Unlike SOM, where the topology of the network (number of units and connections) is predefined, GNG learns the topology by adding both units and connections as more information is received. This, as later shown, can translate to one or more modeled networks of arbitrary shape (as given by the data distribution), where each represents different populated areas in the data. Counting the networks, therefore, provides an estimate of the number of clusters in the dataset, k [20].

Our proposed entanglement prevention seeks to create lower dimensional graph representations of the manifold, which in turn is expected to (a) foster the creation of separate networks whenever suggested by the density distribution of the data, and (b) improve its visual representation in the two-dimensional plane, hence facilitating the visual disclosure of cluster patterns.

2.2. MapReduce

MapReduce is a programming model proposed by Dean and Ghemawat [23] in order to scale computations across distributed infrastructures in a reliable manner. The model is composed of two stages: one which transforms data instances using a *map* function M; and another which aggregates the transformed values using a *reduce* function R. The output cardinality of instances from the first stage is usually equal to the cardinality of the input—sometimes lower if a filtering condition is applied—while the cardinality of the second stage is, as the name implies, reduced.

Fig. 2 gives an example of a MapReduce process where a dataset X is split into four partitions. Each partition is transformed from a type y to a type j using the map function M, and the resulting transformations are then reduced with a function R that aggregates instances by key sets K and T (e.g., years or months). Aggregations by key are common practice, but they can also be done on the entire the dataset (i.e., without a key), hence producing a single result. This is the case, for example, in our proposed

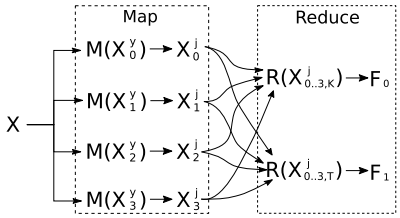


Fig. 2. MapReduce with four partitions.

GNG parallelization. Both stages can be performed in parallel, with the only condition that, in order to have consistent results, the reduce function is both associative and commutative.

Our proposed parallelization of GNG uses the map stage to model each data partition separately, and then the reduce stage to merge each model into a global one. Details are given in Section 5.

3. Related work

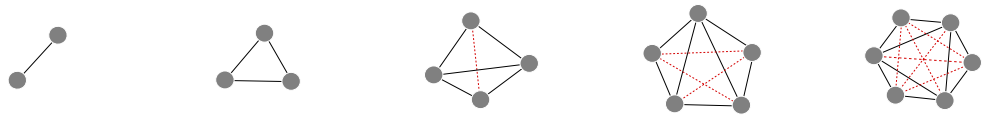
3.1. Visualizing cluster patterns

There are different machine learning (ML) techniques for visualizing cluster patterns, most of which fall under the umbrella of DR. Such techniques seek to project data from an N-dimensional to a Y-dimensional space, where $Y < N$, while preserving, to the largest possible extent, the local or global neighborhood relationships of the original N-space (distance-wise, e.g., Euclidean). If $Y \in \{2, 3\}$ then the projected data can be visually encoded using 2D or 3D scatter plots. Examples of DR techniques are PCA [3], MDS [27], Isomap [28], t-SNE [4] and UMAP [29]. In particular, the latter two have proven to scale well to high-dimensional datasets by modeling the manifold that emerges from local neighborhoods, i.e., by prioritizing the preservation of smaller distances over global ones (see [13] for an extensive benchmark). Unlike GNG's topology models, from where it is also possible to draw an estimate of the number of clusters, DR projected values do not lend themselves to the automated estimation of k.

Other methods for visualizing cluster patterns fall under the *clustering* category. Examples are Ward hierarchical clustering [6], which constructs a tree that can be visually encoded using a dendrogram; OPTICS density-based clustering [7], whose output can be visually encoded using a reachability plot; or the aforementioned neural-based techniques SOM and GNG, which can, respectively, be visually encoded using a U-Matrix [18] or force-directed placement (FDP) [30]. SOM and GNG are similar in the sense that both are topology learning techniques, both work incrementally, and both perform vector quantization. SOM, however, has a static network topology of predefined connections among units that limits its learning to the movement of the prototypes. Despite their quantization and incremental learning benefits, both are likely to score poorly in terms of interpretability and accuracy, as compared to some DR techniques, when used to estimate the number of k clusters in a dataset [19]. This paper intends to improve these aspects of the GNG algorithm, based on a previous implementation from the library proposed in [30] where sampling is used to collect batches of the distributed data into a single machine and run per-batch epochs of Fritzke's algorithm.

3.2. Scaling the growing neural gas

A type of GNG disentanglement was suggested by Costa and Oliveira [20], where they rank and eliminate edges after training using a three-step strategy. First, they duplicate the trained model, remove all edges, and create new ones by resubmitting the data.



(a) 1D. (b) 2D. (c) 3D. (d) 4D. (e) 5D.

Fig. 3. Graph structures of different dimensionalities. Red dashed lines are examples of connections which could be removed in order to lower the dimensionality of a graph.

During resubmission, a utility variable is associated with each edge (which is linearly increased during step 7), and only the edge-related steps—i.e., steps 2, 3, 7, and 8—are executed. The second step is to remove edges that are not at the intersection of both the first model and the duplicated model. Finally, the last step removes other edges whose utility value, or length (based on the Euclidean distance between the prototypes of the units they connect), does not meet or exceeds a certain threshold. Unlike their solution, which carries a post hoc removal of edges, the mechanism proposed in this paper avoids entanglement during the learning process itself. This has an impact on the movement of the units during training, which influences the accuracy of the model (in terms of MSE) as well as the number of created networks.

Concerning the improvement of execution times, Fliege et al. [31] proposed a MapReduce version of GNG following the batch parallelization approach suggested by [32]. Namely, their solution creates an initial model with two connected units (as in the original algorithm) that is then broadcast to each of the working threads (or nodes in the case of a cluster infrastructure). A *map* function then fits each model to a different partition of the dataset by executing steps 2 to 6, i.e., by moving units in the data space and without making changes to the graph. A *reduce* function then merges the fitted models into a global one, by averaging the movement of the prototypes, the units' errors, and the ages of the edges. Finally, the resulting model is updated by executing steps 7 to 10, i.e., by adding and removing edges, and by adding a new unit (if the maximum number of units has not been reached). The process repeats itself until a convergence criterion is met. Their experiments report faster execution times, but do not report accuracy. The MapReduce version proposed here follows another parallelization approach suggested by [32], where separate GNGs are fitted to different data partitions by executing steps 2 to 10 (hence, letting each grow separately) and then merging them by using the prototypes in one GNG as the training data for another GNG (more details are given in Section 5). The advantage of this approach is that bigger training steps are taken, since more than one unit is added per epoch. We elaborate more about our approach and theirs in Section 8.

Orts et al. [33] proposed a CUDA version of GNG. Their solution parallelized the second step of the learning process, i.e., finding the two closest units to an input signal, as it was found to be the most computationally expensive part; the remaining steps still run in a sequential manner. This step, however, as shown in their experiments, is only worth parallelizing when the number of units used in the modeling is greater than a given threshold ($>$500); GPU parallelization for a lower number of units would not show significant improvement (or could even hinder) the execution times due to the transfer latency with the CPU. In the application area of 3D reconstruction—to which they targeted their contribution—such performance improvements are helpful since it is common to have over 500 units. This, however, is not the case for visualizing cluster patterns, where the number of units is sought to be minimized in order to avoid overplotting and visual cluttering, and reduce the computational burden of user interactions. Compared to their work, we propose a general solution (not GPU-specific) to parallelize the entire pipeline of the learning process (not just one step), and our solution also deals with datasets that are geographically distributed. These two solutions are, nevertheless, not mutually exclusive; it would be possible to parallelize the entire pipeline so that GNG is executed on different CPU threads or worker nodes, while also parallelizing GNG's second step on GPU threads in each worker node.

Zhu et al., [34] proposed the use of random projection, a feature extraction technique, as a pre-processing step to make GNG a viable technique in the context of the specific challenges of data streaming. Random projections are used by the authors to reduce the dimensionality of a dataset and, hence, lower the execution times of GNG while "still finding a solution which is related to clustering in the original high-dimensional space". This is similar to the use of PCA in [4] where t-SNE is presented, but in the context of streaming data and clustering with GNG. Random projections, however, have also been used as a pre-processing step for scaling clustering algorithms that work on static data [35], which suggests that they can also be used as a pre-step to the enhanced GNG version proposed here and, hence, further decrease its execution times.

4. Avoiding entanglement

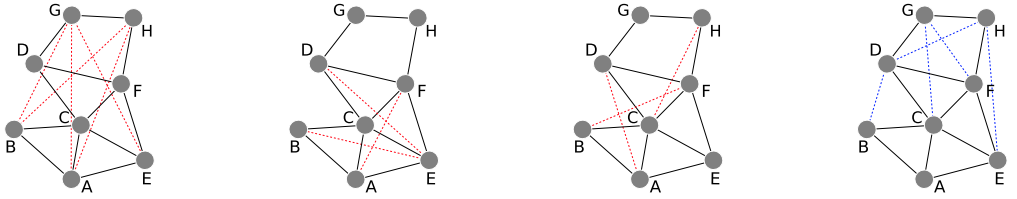
This section describes the first proposed method, aimed at providing more meaningful and accurate cluster embeddings of GNG's modeled topologies. Fig. 1 showed two projections of the MNIST dataset (details of the dataset are given in Section 6), one using force-directed placement on a GNG model (left), and the other a scatter plot of a 2D t-SNE projection (right). In the former, each node represents a GNG unit, with nodes sizes encoding density (i.e., the number of data points a unit represents), and the length of the edges encode the Euclidean distances between units' prototypes.

It is possible to see, in Fig. 1, that GNG created a highly connected network that hardly reflects any of the cluster patterns that other techniques, such as t-SNE, manage to disclose. A possible explanation for this behavior is that the connections created by GNG form high-dimensional graph structures. Fig. 3 illustrates the idea of how graph structures grow in complexity as they increase in dimensionality. In fully connected networks, the more units, the higher the dimensionality of the graph, and, hence, the more overlapping edges in the 2D plane.

The original GNG has no restrictions on the dimensionality of the graph structures it creates. Namely, step 7 of the GNG will connect the two closest units (μ_a and μ_b) to a given signal (δ), regardless of where they currently lie within the network. This behavior is not necessarily *wrong*, since it reflects the complexity of relationships in higher-dimensional spaces. Still, it might be undesired for the tasks of cluster analysis and visualization.

An approach to fostering the creation of lower-dimensional graph structures, and avoiding overlapping edges in the embedding, is to restrict the creation of connections among units. Our proposed solution does this by requesting five conditions to be met in step 7 when units μ_a and μ_b are not already connected. Namely, μ_a and μ_b may connect, if and only if:

1. They are second-degree neighbors, i.e., they have at least one neighbor in common—a *bridge*.
2. They have no more than two bridges.
3. There are no connections among bridges (if there are two).



(a) 1st cond. (b) 3rd cond. (c) 4th cond. (d) Allowed.

Fig. 4. Examples of forbidden and allowed connections.

4. If there is only one bridge, then μ_a and μ_b should have no more than 1 common neighbor with it. That is, if μ' is the bridge between μ_a and μ_b, then neither μ_a nor μ_b should have 2 or more common neighbors with μ'.
5. The second winning unit (μ_b) belongs to a *small* network.

The first condition avoids *curling* the network from its current representation of the manifold, by not allowing distant units (graph-wise) to connect. In Fig. 4a, for example, unit A would not be allowed to connect to units G and H since there are no common neighbors between them. According to the first condition, unit A may connect to units D and F since they have B, C, and E as common neighbors; the second condition, however, would not allow these connections to take place. Other forbidden connections of this rule are shown with red dashed lines.

Conditions 2 to 4 aim to avoid local graph structures from becoming high-dimensional, or, simply put, from creating overlapping edges in the two-dimensional plane. According to condition 3, a connection between B and E is not allowed (see Fig. 4b) because their common neighbors, C and A, are connected—hence avoiding edge $\overline{BE}$ from overlapping $\overline{CA}$. Condition 4 (see Fig. 4c) also seeks to avoid overlapping edges while still being permissive, but in slightly more complex situations. This is the case of connections between D and A, and D and B; the latter should be allowed ($\overline{DB}$ would not overlap any other edge), while the former should not ($\overline{DA}$ would overlap $\overline{BC}$). The 4th condition resolves this by checking that their bridge C has no more than *one* common neighbor with either of the units in question. Connecting D and A is forbidden, for example, because their bridge C has *two* common neighbors with A, i.e., nodes B and E. The 2nd condition (i.e., no more than two bridges) simply disregards more complex cases, which are more likely to create overlapping edges in the two-dimensional plane. The allowed connections so far are shown in Fig. 4d. It is possible to see that, despite the former restrictions, higher-dimensional structures can still be constructed (see, for example, that adding $\overline{CG}$ would overlap $\overline{DF}$). Such connections could be avoided by checking not just common neighbors between a bridge and the connecting units, but *paths* which may get on the way. Checking if such paths exist, however, would make the training quite computationally expensive, which is why it is not suggested as part of the solution. Moreover, the results from the tests described in Section 7 show that these conditions alone make considerable improvement over the state of the art.

The final condition aims to allow small satellite networks, i.e., networks whose units are close to other networks in the Euclidean space, to reconnect to other networks. The first condition is *strict* in the sense that it does not allow separate networks to reconnect. Several trials showed that this is the desired behavior when it comes to *reasonably* sized networks, e.g., with more than 2 or 3 units. Smaller networks, on the other hand, may be the result of a split early in the training, when a significant portion of the population has not yet been seen. The third condition gives small networks the opportunity to reconnect to others while also allowing them to become independent.

5. Parallelizing GNG

This section describes the second proposed method, which aims at enabling GNG's execution over distributed infrastructures while leveraging their computational resources. The sequential version of GNG can be defined as

$$L(G_0, X, e) \rightarrow G_e^X \tag{1}$$

where L is GNG's learning algorithm, which fits an initial model G_0 to a training dataset X, and produces a trained model G_e^X after e epochs. The proposed parallel version uses the MapReduce programming model to split the learning process into r partitions of X (where $r > 1$, ideally tailored to the computational resources and the size of the dataset) and then merge the results from each partition into a single model.

Concretely, the first step to parallelize GNG is to split the dataset X into r partitions $X^i, i \in [1, ..., r]$, and then apply a map function (M) that partially fits (with a single pass) separate GNG models to each of the partitions X^i. Data partitioning should be balanced (i.e., each partition should have a similar number of data points) so that each parallel process takes similar time, and preferably random (i.e., data points randomly assigned to each partition) to avoid bias in the learning (this is, nonetheless, mitigated in the merging stage as later explained). Assuming that X has been split following these conditions, the map function is defined as follows

$$M(G_t, X^i) \rightarrow L(G_t, X^i, 1) \rightarrow G_{t+1}^i \tag{2}$$

That is, M is a function that applies a one-pass of L to the i^{th} partition of X, at a given epoch t. The first epoch ($t = 0$) looks as follows:

$$M(G_0, X^0) \rightarrow G_0^0$$
$$M(G_0, X^1) \rightarrow G_0^1$$
$$\cdots$$
$$M(G_0, X^r) \rightarrow G_0^r$$

This results in r partially fitted models, one per partition. These models are then merged using the following reduce function:

$$R(G_t^i, G_t^j) \rightarrow L(G_t^i, P_t^j, 1) \tag{3}$$

where G_t^i and G_t^j are the models fitted to partitions i and j, and P_t^j are the units' prototypes in model G_t^j. The merging mechanism is simple in the sense that it reuses GNG's learning logic. Namely, the prototypes of model G_t^j are used as a training set for a one-pass run of GNG, using model G_t^i as the initial state. In other words, G_t^i is partially fitted to G_t^j's learned space. The merging order is preferably random to prevent bias. This analogous to counterbalancing in statistical experiments, where each participant (GNG learner) is shown different conditions (partitions) in a different order.

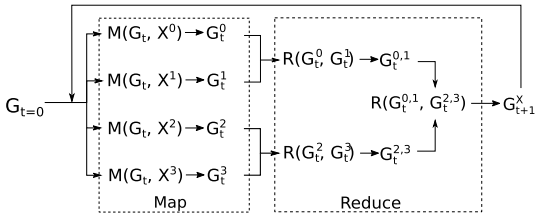


Fig. 5. Parallel GNG learning example with four partitions.

The resulting model from the reduce stage is then used as the initial state for the mapping stage in the following epoch. Fig. 5 exemplifies the overall process for a dataset split into four partitions. The hyper-parameters for both the map and the reduce stages can be the same as the ones used for the sequential GNG version.

All parallel versions assessed in this paper are based on this model. The only difference in s-GNG is that it samples each data partition during the map stage. Concretely,

$$M(G_t, X^i, s) \rightarrow L(G_t, Sample(X^i, s), 1) \tag{4}$$

where *Sample* is a uniform sampling function, and s states the fraction size (in percentage) that is to be sampled from the given set of points. Sampling is expected to reduce training times and avoid noise from the data, and while it may arguably lead to a loss in accuracy, our empirical results show that this is not the case.

6. Experimental setup

This section describes the metrics used to assess performance, the independent variables (i.e., GNG versions and datasets), and the conditions under which the experiments were carried.

6.1. Performance metrics (dependent variables)

Five performance metrics were used, all measured on a per-epoch basis: execution time, mean squared error (MSE), class separation error (CSE), number of created networks (CN), and the number of redundant networks (RN).

MSE measures the average squared difference from each data point to its closest unit:

$$MSE = \frac{1}{N}\sum_{i=1}^{N} MinSE(G^X, x_i) \tag{5}$$

where N is the size of the dataset, and $MinSE$ a function that returns the minimum squared error between data point x_i and each of the units' prototypes of model G^X. This metric measures the quality of the model in terms of how well the units represent the distribution of the data in the multidimensional space. The lower the number, the better.

CSE, on the other hand, measures the goodness of the connections among units with respect to the class they represent. This is quantified in terms of the percentage of edges that are connecting units that represent two different classes, i.e.

$$CSE = \Gamma / E \tag{6}$$

where E is the total number of edges, and Γ the number of edges connecting units of a different class. Class representation, in this case, is given by a majority vote from the data points a unit represents. That is, the most frequent label in the subset of X, which is closest to a given unit (in terms of Euclidean distance). As in MSE, the lower the value the better.

The last two metrics are CN and RN. The former is a simple count on the number of independent networks in the model, while the latter is a count of networks that are redundant in terms of the class they represent (i.e., the number of networks representing the same class). Similar to the class of a unit, the class of a network is given by a majority vote from its units. Greater CN values, together with lower RN values, mean a closer representation of the true number of classes in a dataset, i.e., closer to the ground truth.

Table 1
Datasets.

	Dataset	Size	Features	Labels
S	Dermatology	366	34	6
	BreastCancer	569	9	2
	Abalone	4K	7	29
M	Isolet	6.2K	617	26
	MNIST	60K	784	10
	FashionMNIST	60K	784	10
L	E-MTAB-6169	787K	50	–
	Glove	400K	200	–
	GoogleNews	3G	300	–

6.2. GNG versions and datasets (independent variables)

Five versions of GNG were used to assess the validity of the three proposed changes, i.e., entanglement avoidance, parallelization, and parallelization with sampling. Namely:

- GNG: Sequential GNG (i.e., original version used as a baseline).
- u-GNG: Sequential and untangled GNG.
- parGNG: Parallel GNG.
- u-parGNG: Untangled and parallel GNG.
- s-GNG: u-parGNG with sampling.

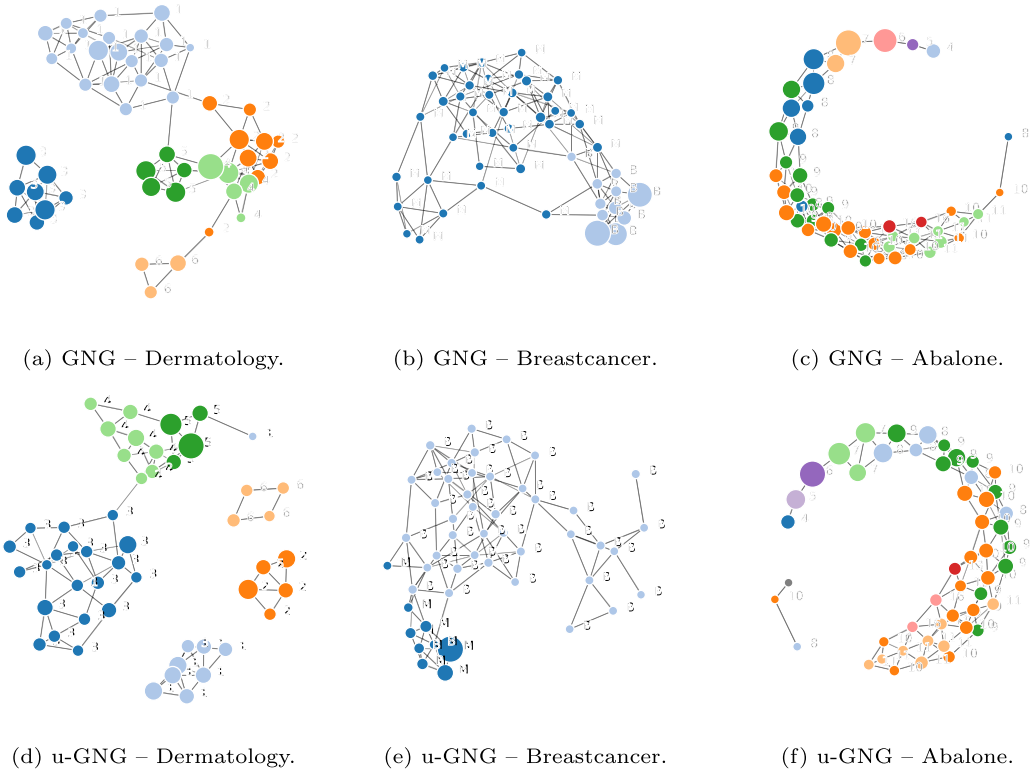
Six datasets were used in the evaluation (see Table 1): 3 *smaller* (S) datasets, with less than 5K data points, used only for testing the sequential versions (i.e., GNG and u-GNG); 3 *medium* (M) sized datasets, between 6K and 100K data points, used to test all five versions; and 3 *larger* (L) datasets, with over 400K datapoints, assessed using parGNG and s-GNG projections. These latter datasets do not have labels, and thus, their performance is measured only in terms of MSE and CN.

The S datasets are Dermatology, comprised of skin sample measurements associated with erythemato-squamous diseases; BreastCancer [36], comprised of benign and malignant cell nuclei measurements; and Abalone [37], comprised of measures of mollusks of different ages. These were all taken from the UCI machine learning repository [38]. The M datasets are Isolet [39], comprised of spoken letters of the English alphabet (also taken from UCI); MNIST [21], comprised of 28×28 pixel images of handwritten digits; and FashionMNIST [40], comprised of 28×28 pixel images of clothing articles. Finally, the L datasets are E-MTAB-6169 [41], comprised of chicken transcriptomic profiles; Glove [42], comprised of word embeddings from Wikipedia pages; and GoogleNews [43], comprised of word embeddings from news articles.

6.3. Implementation and parameters

All GNG versions, as well as metric functions, were implemented using the Scala programming language. The Apache Spark library was used to implement the parallel versions, thus leveraging from its abstraction of the MapReduce model. A public implementation of these versions is provided as an off-the-shelf Visual Analytics library [44] here: https://github.com/eliovr/visualgng.

The environment where the sequential versions were tested was a 64 bit Ubuntu 18.04 computer, with 62.4Gb of RAM and Intel Xeon E5/Core i7. The environment for the parallelized versions, on the other hand, was a three-node cluster (two workers



(a) GNG – Dermatology. (b) GNG – Breastcancer. (c) GNG – Abalone.

(d) u-GNG – Dermatology. (e) u-GNG – Breastcancer. (f) u-GNG – Abalone.

Fig. 6. Embeddings of the **S** datasets. Training for these was carried over 50 epochs with a maximum of 50 units. Colors and text represent classes in the data. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 2
Experimental settings. *8 partitions were used with the parallel GNG versions, and 1 for the sequential. **24 partitions were used for the GoogleNews dataset.

	S	M	L
Max. units	50	120	200
Epochs	50	100	100
Partitions	1	1, 8*	8, 24**
Scaling	Yes	No	No

and a driver), each using Ubuntu 18.04, 4 VCPUs, and 4Gb of RAM. The aim of this setup is to test and showcase the parallel versions of GNG in a real distributed environment. Here, datasets were distributed using Hadoop HDFS. For the GoogleNews dataset, however, the first environment was used since it provided 24 cores for parallel executions—the distributed environment provided only up to 8 effective threads (2 workers $\times$ 4 VCPUs).

The original GNG hyper-parameters were, in all cases, the same as those suggested by Fritzke [16] (as in [45]). Concretely: $\epsilon_b = .2$ (adaptation step of the closest unit), $\epsilon_n = .006$ (adaption step of the neighbors), $\lambda = 100$ (frequency with which units are added), $a_{max} = 50$ (maximum edge age), $\alpha = .5$ (error reduction rate of the neighbors of a new unit), $d = .995$ (error reduction rate of all units). Other parameters, such as the maximum number of units and the number of epochs, were set based on the size of the datasets (see Table 2).

The two proposed methods, along with the proposed sampling, include three additional parameters: the size of small networks in entanglement prevention (condition 5); the number of partitions r in parallel learning; and the sampling factor s in parallel learning with sampling. The former was set to 3 (i.e., networks of 3 units may reconnect), the sampling factor was set to 0.5 (i.e., sample fifty percent from each partition), and the number of partitions r was set to 8. In the case of GoogleNews, r was set to 24 in order to leverage from all computational threads—as previously mentioned in the environment settings.

S datasets were scaled to a one-unit variance since their original features had different scales. Finally, 10 test trials were carried in all cases. This allows assessing the variance in the measurements that would result from the instantiation of the initial models (G_0), the merging order during the reduce phase, as well as the sampling used in s-GNG.

7. Results

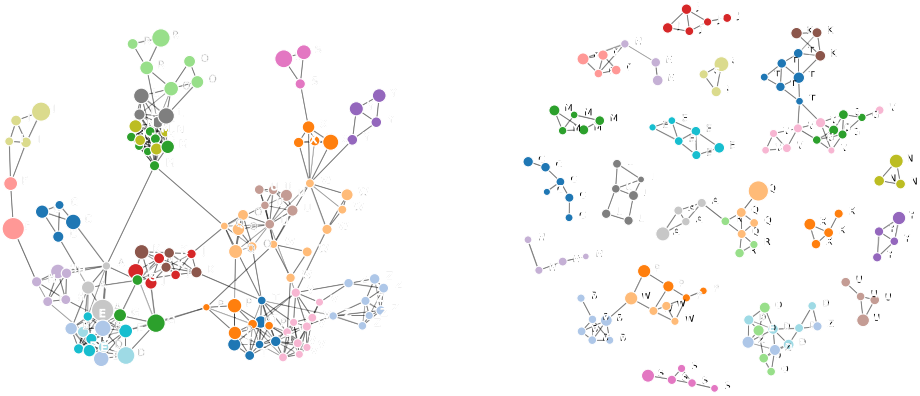
The results are described in two parts: first, in terms of example embeddings (2D visual representations) of the S, M, and L datasets using different GNG versions; and second, in terms of the performance metrics.

7.1. Embeddings

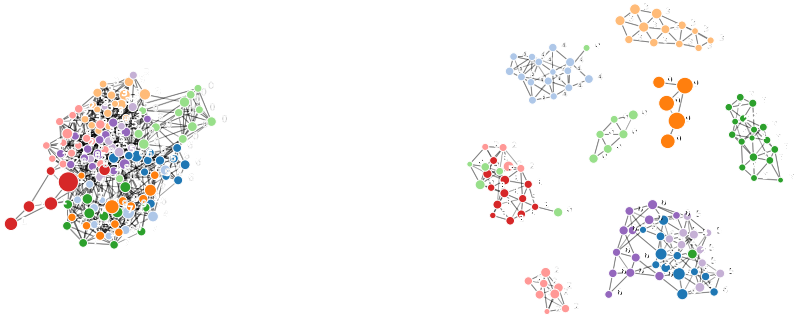
Examples of the embeddings resulting from the different GNG versions are shown in Fig. 6 (for the S datasets), Fig. 7 (for the M datasets), and Fig. 8 (for the L datasets). For the M datasets, only embeddings from GNG and s-GNG are shown; the others (u-GNG, parGNG, and u-parGNG) are provided in the supplementary material. Figs. 6 and 7 use color encodings and tags to identify different classes given by the majority votes (as explained in Section 6).

In Fig. 6, not much difference can be seen between the GNG and u-GNG embeddings of the S datasets, except for Dermatology dataset. In this latter case, GNG created 2 networks, one representing the 3s and the other the rest, while u-GNG created 4 networks, three of which present purity (i.e., all units in a network represent the same class) and a fourth one with a mixture of the remaining three classes.

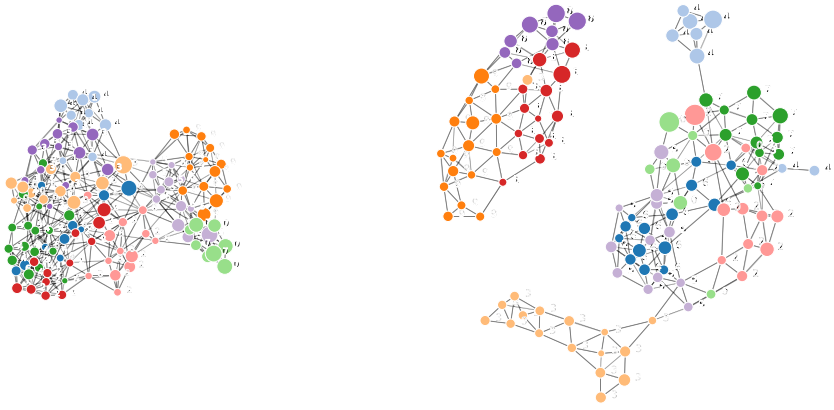
Embeddings of M datasets, on the other hand, show larger differences between the results of GNG and s-GNG (see Fig. 7). Here, GNG created a single network for all datasets, with particularly



(a) GNG – Isolet. (b) s-GNG – Isolet.



(c) GNG – MNIST. (d) s-GNG – MNIST.

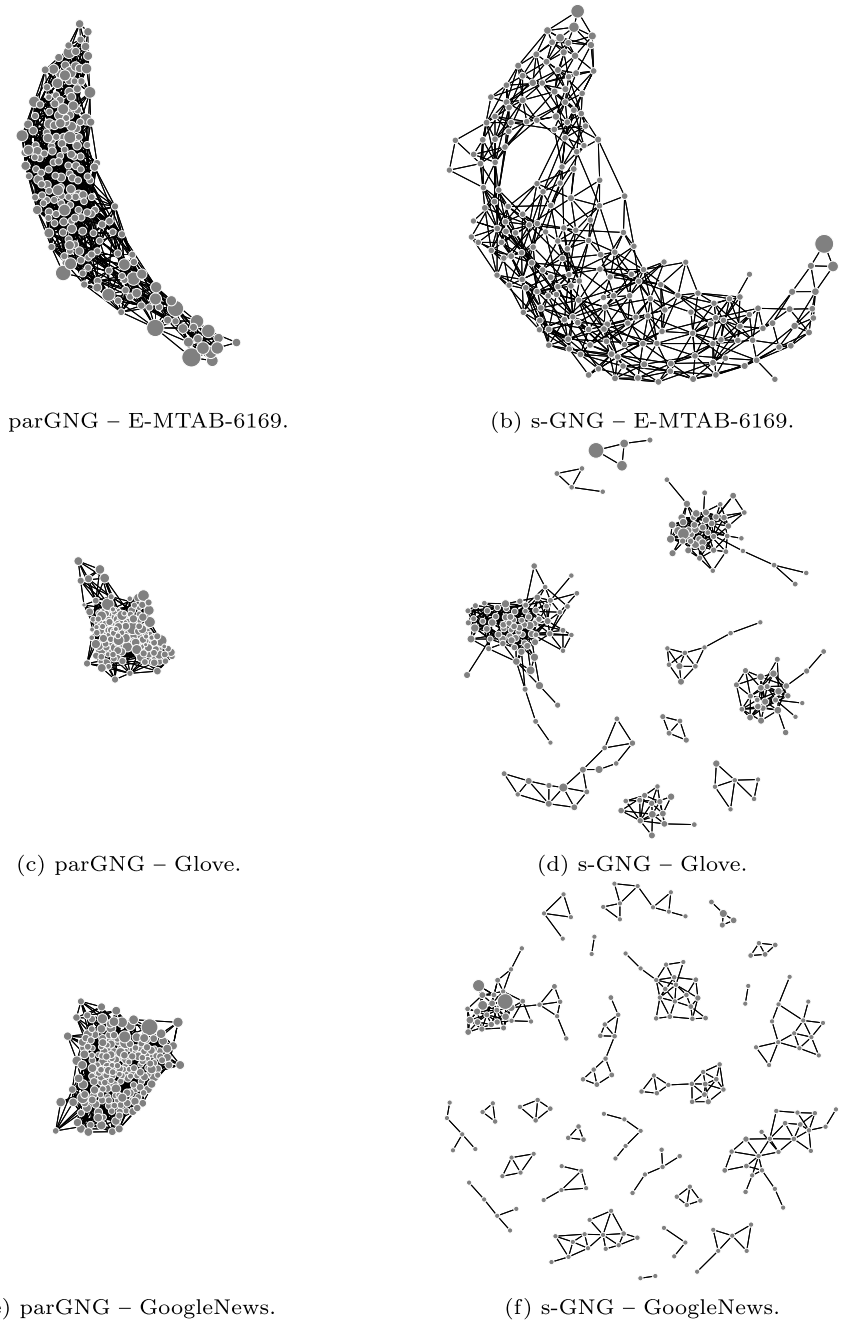


(e) GNG – FashionMNIST. (f) s-GNG – FashionMNIST.

Fig. 7. Embeddings of the **M** datasets. Training for these was carried over 100 epochs with a maximum of 120 units. Colors and text represent classes in the data.

cluttered results on the image datasets (i.e., MNIST and FashionMNIST). s-GNG, on the other hand, created different separations (independent networks) for each dataset: 17 networks (12 being pure) out of 28 classes for Isolet; 8 networks out of 10 classes for MNIST (of which 6 are pure, and one with a biased majority towards 3s); and 2 networks out of 10 classes for FashionMNIST (with mixed class representations, but arguably with much less overlapping elements and visual clutter). Other GNG versions (provided in the supplementary material) show slightly different results. parGNG had similar outcomes compared to GNG on the images datasets (i.e., a single clutter network), but produce different independent networks for Isolet (10 networks, 5 of which are pure). u-GNG and u-parGNG, on the other hand, produced models with a few differences when compared to s-GNG, i.e., the former did not make a clear separation between 5s and 3s, while the latter did not make a clear distinction of the 8s. These differences, however, as previously mentioned, can vary slightly from one run to another.

Finally, Fig. 8 shows the differences in the performance between parGNG and s-GNG on L datasets. In the case of E-MTAB, both versions produced similar results: a single, elongated network, with thinner extremes and a wider set of interconnected units in between. Units in s-GNG, however, are more spread and, therefore, less cluttered due to a lower number of connections. For the word embedding datasets (Glove and GoogleNews), parGNG produced a single and highly connected network (similar to the image datasets) which resulted in highly cluttered representations. s-GNG, on the other hand, produced several independent networks, of which many have low dimensionality.



(a) parGNG – E-MTAB-6169. (b) s-GNG – E-MTAB-6169.

(c) parGNG – Glove. (d) s-GNG – Glove.

(e) parGNG – GoogleNews. (f) s-GNG – GoogleNews.

Fig. 8. Embeddings of the **L** datasets. Training for these was carried over 100 epochs with a maximum of 200 units.

7.2. Performance metrics

Fig. 9 shows the execution times per epoch, for all GNG versions on the M and L datasets. Execution times on the S datasets are provided in the supplementary material. The table is divided into M and L datasets, and into sequential and parallel versions.

It is possible to see that, for the M datasets, parallel versions had lower training times per epoch than the sequential versions. It is also possible to see that, in most cases, the untangled versions (u-GNG and u-parGGN) achieved lower execution times than the original GNG (i.e., GNG and parGNG), with the exception of Isolet (in both sequential and parallel versions) and FashionMNIST (in the parallel version), where the difference is not as clear. In the case of the S datasets, the untangled and original GNG versions (both sequential) had similar performances.

Table 3
Execution time ratios, i.e., how many *times* faster (in average) is one version (e.g., parGNG) compared to another (e.g., GNG).

	Isolet	MNIST	F-MNIST	E-MTAB	Glove	G-News
GNG vs. parGNG	9.5	13.6	13.4	–	–	–
u-GNG vs. u-parGNG	8.8	7	6.2	–	–	–
u-parGNG vs. s-GNG	2.4	1.9	1.9	2	1.9	2

Table 3 shows the average execution time ratios (i.e., how many times faster is one version compared to another). The first two rows represent a sequential GNG version versus its analog parallel version. The last row, on the other hand, compares a parallel ex-

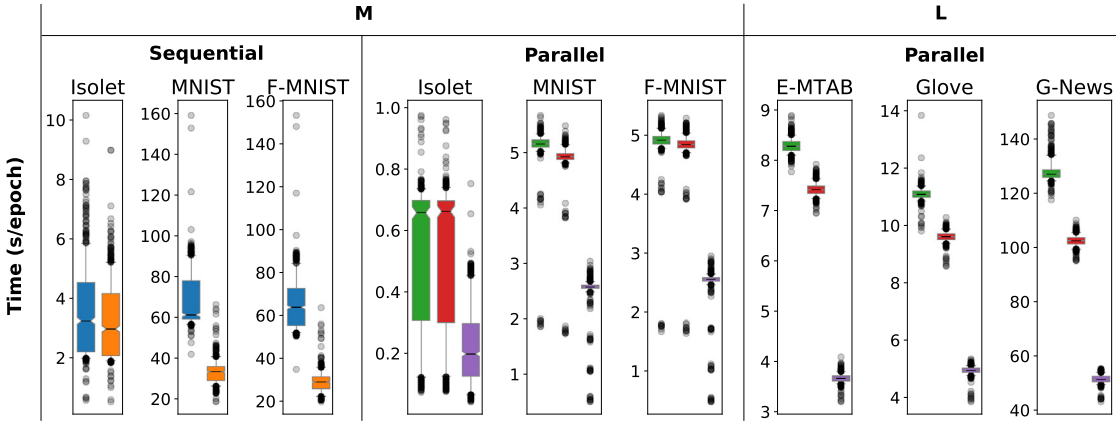


Fig. 9. Execution times on the M and L datasets. Colors correspond to GNG, u-GNG, parGNG, u-parGNG, and s-GNG. G-News stands for GoogleNews.

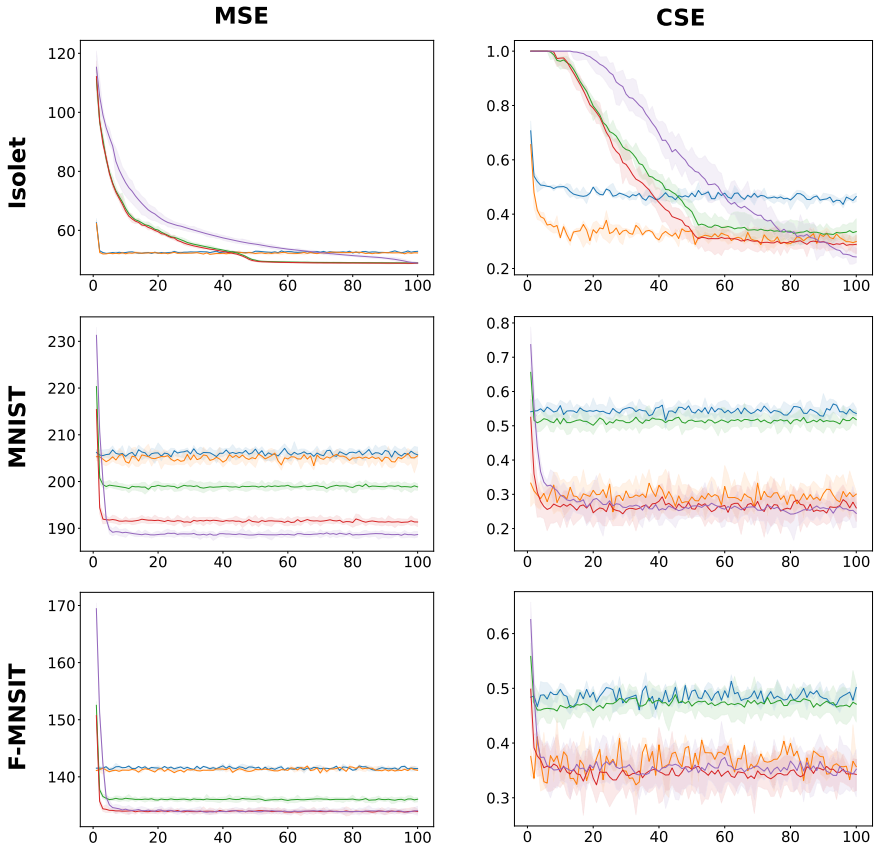


Fig. 10. MSE and **CSE** performance results on the **M** datasets. Series represent GNG, u-GNG, parGNG, u-parGNG, and s-GNG. Solid lines depict the mean values per epoch from the ten trials, while lighter areas the minimum and maximum values.

ecution to a parallel execution *with sampling*. It can be seen that parallel versions performed up to 13 times faster than sequential ones, and that the performance gain from the *sampling* version seems to be correlated to the sampling factor (i.e., 0.5).

Results for the other metrics are shown in Figs. 10, 11 and 12. The former two provide the results on the M datasets in terms of MSE and CSE, and in terms of CN and RN, respectively. The latter, on the other hand, shows the MSE and CN results on the L datasets (these do not have CSE, CN, nor RN metrics since they do not have classes). Results on the S datasets are provided in the supplementary material, since there are no large differences between the performances of GNG and u-GNG. For all these figures, each column represents a metric, each row a dataset, and each series within each plot a GNG version. The solid lines in each plot depict the mean value of each metric per epoch, while the light areas show their minimum and maximum values. The horizontal red lines in the CN plots represent the ground truth, i.e., the number of labels in each dataset.

In terms of MSE, u-GNG performed similarly to GNG on both the S and the M datasets. Interestingly, parGNG performed better than GNG and u-GNG on both of the image datasets (this is discussed in Section 8), while u-parGNG and s-GNG had the best performance. In the case of the L datasets, both untangled versions (u-parGNG and s-GNG) performed better than parGNG, with the exception of GoogleNews where there is no clear difference in performance concerning MSE.

CSE (class separation error) results, on the other hand, show that GNG and u-GNG had similar performance on the S datasets

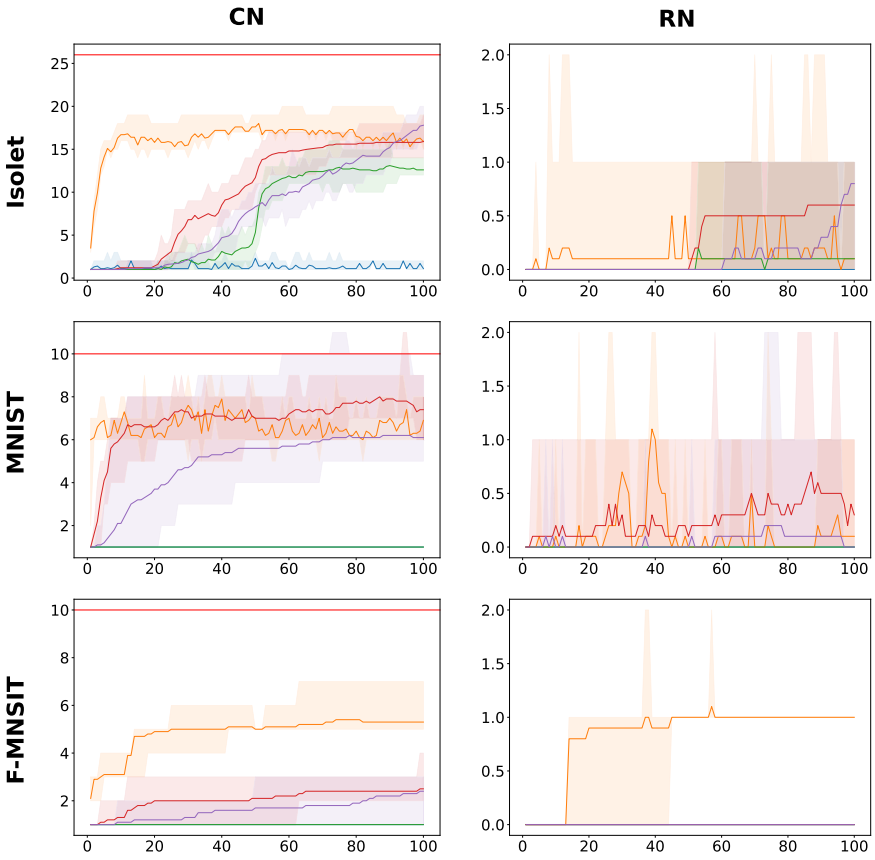


Fig. 11. **CN** and **RN** performance results on the **M** datasets. Series represent GNG, u-GNG, parGNG, u-parGNG, and s-GNG. The horizontal red lines in the CN column represent the ground truth, i.e., the number of labels in the corresponding dataset.

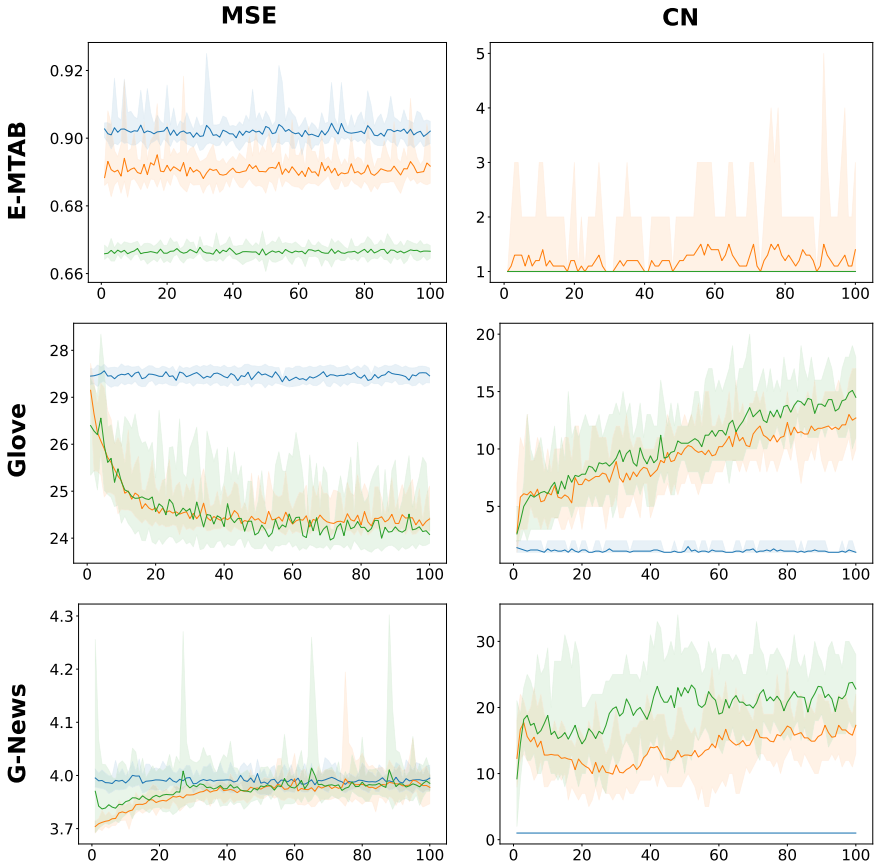


Fig. 12. **MSE** and **CSE** performance results on the **L** datasets. Series represent parGNG, u-parGNG, and s-GNG.

(a) MNIST. (b) FashionMNIST

Fig. 13. s-GNG projections of the image datasets, with images rendered from each unit's prototype.

and that u-GNG outperformed, i.e., had lower values than GNG, on the M datasets, with the most significant improvements seen again on the image datasets. Isolet shows to be a particular case where the parallel versions took a longer time to achieve lower CSE values than the sequential versions. This is further discussed in Section 8.

Concerning CN, all untangled versions (u-GNG, u-parGNG, and s-GNG), for most datasets (with the exceptions of BreastCancer and Abalone) modeled a higher number of networks than GNG. It is possible to see that the number of created networks varied from one run to another, and that such variance differs from one dataset to another. s-GNG, for example, had the highest mean variance when applied on the MNIST dataset, but had similar values on Isolet and FashionMNIST.

Finally, RN shows that between 0 and 2 redundant networks were created during training, particularly in the cases of the M datasets. In those cases, the metric serves more as a benchmark between untangled versions, since the original GNG only created one network (with the exception of Isolet where parGNG created between 10 and 15 networks).

Overall, untangled versions performed equal or better than the original GNG in terms of MSE, while modeling a higher number of networks (CN), with some redundancy (RN), but still achieving better cluster separations (CSE)—specially in the image datasets. And parallel versions performed equal or better than sequential versions in terms of MSE and CSE, but with lower execution times.

8. Discussion

8.1. Avoiding entanglement

As seen in the results, both in the projections and the CN plots, entanglement prevention is better perceived in larger and high-dimensional datasets, as in the cases of the image and word embedding datasets. There, the connectivity restrictions fostered the creation of both independent networks as well as lower-dimensional graph structures. The latter is particularly well represented by the networks of Qs and Us in Isolet (Fig. 7b), by the networks of 5s and 6s in MNIST (Fig. 7d), by the region of 8s (bags) in FashionMNIST (Fig. 7f), and by the small networks in Glove and GoogleNews (Figs. 8d and 8f). These show triangular connections with few overlapping edges.

The quantitative results show that, in general, MSE accuracy was preserved in the obtained GNG models, even when simplified and lower-dimensional networks were enforced. Since the main goal of entanglement prevention is not to improve accuracy *in terms of MSE*, but to foster the creation of 2D graph structures so that more accurate and meaningful *cluster representations* can be constructed, the results were promising.

The obtained graph structures, however, were not always 2D, as is reflected by the overlapping edges in some parts of the embeddings (in the same figures). This is exemplified by the network of 4s, 7s and 9s in MNIST, the region of 2s (pullovers) and 4s (coats) in FashionMNIST, the larger networks in Glove, and in the whole E-MTAB network (see Fig. 8b). Such results might reflect the complexity of neighborhood relations in some data spaces as compared to others. For example, the space covered by units representing 1s (in MNIST) can be regarded as a *simple* one, since fewer units and edges were required to represent it. This is better perceived in Fig. 13, where the units' prototypes have been rendered into images. There, it is possible to see an almost linear change in values that goes from fully standing 1s (on the upper end of the network), to the most tilted 1s (on the lower side), with almost no forking paths (i.e., other variations of one) in between. A complex space, on the other hand, can be exemplified by that of handwritten 4s, 7s, and 9s, where more units and edges were needed to model the distinct ways in which these have been written. In such a case, the gradual shift in values happened both within and between classes, with several forking paths and no clear gaps between classes.

The differences between such simple vs. complex relations are further exemplified by s-GNG's representation of FashionMNIST. In Fig. 13b, the network of 'Trousers', at the uppermost part, may be regarded as a simple space, since it required fewer units and connections to represent it. An example of a more complex space, on the other hand, is that of 'Sneakers', 'Sandals', and 'Ankle boots' (the left-most network), where more units and edges were required to represent it. Such space, however, is arguably not as complex, since the achieved graph structure is largely two dimensional.

Surprisingly, preventing entanglement improved MSE accuracy in some cases. This is probably explained by the fact that more connections mean that each unit has more neighbors and, hence, has a larger influence during training (i.e., by pulling more units during step 6 of the algorithm). Such behavior will not let different parts of the network settle in their local minima.

It is worth noting that MSE and CSE are not necessarily correlated. The reason is that MSE measures the quality of the modeling

in terms of how well units are standing in the multidimensional space, while CSE measures the quality of the connections among units. This means that GNG and u-GNG could have equivalent performance in terms of MSE (i.e., their units are standing in similar places of the space), but their CSE performance could be very different (i.e., the connection among units is different). This is the case of MNIST and FashionMNIST, as shown in Fig. 10, where MSE results are very similar, but CSE results are different.

Another relevant observation is that entanglement prevention did not seek to create separate networks. This was the particular reason for having chosen the Abalone and the E-MTAB datasets, for example. As it is shown by other DR projections [19], instances of both datasets reflect a gradual change from lower to higher feature values—which is to be expected, at least in Abalone, since instances represent the dimensions of mollusks measured at different ages. Such gradual changes are preserved by the untangled versions, despite constraints placed on the creation of edges.

A limitation shown by these results is the variance in the produced models, particularly in terms of the number of created networks. In some cases, such as s-GNG on FashionMNIST, the number of created networks show an uncertainty of only ± 1 networks around a mean of 2 (on the last epoch), which means that at times s-GNG would produce 1 network, and at times 3. This is arguably a low margin of uncertainty, but it is not always the case. A more sensitive example is that of s-GNG on Glove and GoogleNews. There, the uncertainty on the last epoch is ± 4 around a mean of 15, which translates to a wider range of results that can be produced. A possible explanation for such a variance in the number of created networks is that the strictness of the rules, combined with the stochastic initialization of GNG, result in the creation of independent networks at the beginning of the learning process, which are then not allowed to consolidate (merge) with others, even if they represent different regions of the same cluster. An approach to reducing such variance is to set a higher value (e.g., 100) for a_{max} (the maximum edge age) at the beginning of the training, and reduce it to a given minimum value (e.g., 50) as the learning progresses. In this way, GNG would start by modeling the space using a single network, and then split into separate ones as the tolerance on the age of the edges is lowered.

8.2. Parallel training

In regards to parallelism, results consistently show equal or improved performance both in terms of MSE and CSE, even when using sampling. This points to the possibility of implementing systems that can potentially provide feedback to the users under 10 seconds, even in very large and distributed datasets. This is, of course, always dependant on the resources at hand.

Interestingly, in some cases the parallel versions helped to improve performance in terms of MSE, as shown in the M datasets. This is possibly due to having different learners modeling different spaces, which avoids bias towards the order in which the data points are laid out in the dataset. That is, units might shift from one end of the space to another, depending on the order in which data points are given to the learner. If, for example, data points are ordered in an ascending manner by their L-norm, then GNG will tend to retain a better representation of the spaces with higher values, thus neglecting the spaces with the lower values.

Another interesting case is that of Isolet, where parallel versions of GNG took longer to reach lower MSE and CSE values. The reason for this is the size of the dataset which, in spite of being placed within the M category, is still small compared to the others. The speed with which a model grows is given by λ, the parameter that states the frequency with which units are created. In the case of the parallel versions, this frequency is the same for each learner in the mapping stage, but the number of data points seen by each is fewer, which means that the number of created units per epoch depends on both λ and the number of seen data points. In the case of Isolet, each learner trained on less than 800 data points, which means that, for each epoch, each model in each partition would grow only by 7 units, hence taking roughly 18 epochs to reach the maximum number of units (120). This, however, would be the case if no units are lost in the process, which is the case when units become orphans (i.e., with no neighbors) when connections are removed. Based on this observation, approaches such as the one by Fliege et al. [31], where units are added on a per-epoch basis, would arguably lead to longer execution times when compared to our solution. In terms of accuracy, however, it is unclear at this point which approach is superior. One could, nevertheless, envision a mixed approach, where the initial training epochs are performed by our proposed method until the maximum number of units is reached, and then apply their approach for the subsequent epochs. This may help increase the speed, accuracy and stability of the modeling. An empirical evaluation concerning this is planned for future work.

9. Conclusions

This paper proposed two methods for scaling GNG in the context of visual cluster analysis of large, distributed, and high-dimensional datasets. The first method improves the modeling performance and the meaningfulness of the cluster pattern representations generated by GNG by restricting the created connections during training and, as a result, by creating lower-dimensional graph structures that reduce overplotting and cluttering in the two-dimensional plane. The second method parallelizes GNG's learning process, leveraging from the computing power of distributed infrastructures—or local components—for parallel computing, thus lowering the execution times of the learning process. Improvements such as this facilitate the development of interactive applications that comply with usability principles, such as providing visual feedback under 10 seconds.

The proposed methods were assessed on 9 different datasets, in terms of four quantitative metrics (MSE, CSE, CN and RN), as well as qualitatively through examples of their visual representations. The results show that the first method, regarded as entanglement prevention, improves GNG's accuracy on high-dimensional datasets, as well as the meaningfulness of its cluster pattern representations. In turn, the second method enables the use of GNG in distributed settings, while preserving (or sometimes improving) the accuracy and meaning of the first one.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.bdr.2021.100254.

References

[1] M. Goebel, L. Gruenwald, A survey of data mining and knowledge discovery software tools, SIGKDD Explor. Newsl. 1 (1) (1999) 20–33, https://doi.org/10.1145/846170.846172.
[2] J.W. Sammon, A nonlinear mapping for data structure analysis, IEEE Trans. Comput. C-18 (5) (1969) 401–409, https://doi.org/10.1109/T-C.1969.222678.
[3] H. Hotelling, Analysis of a complex of statistical variables into principal components, J. Educ. Psychol. 24 (6) (1933) 417.
[4] L. van der Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (Nov) (2008) 2579–2605.

[5] E. Becht, L. McInnes, J. Healy, C.-A. Dutertre, I.W.H. Kwok, L.G. Ng, F. Ginhoux, E.W. Newell, Dimensionality reduction for visualizing single-cell data using UMAP, Nat. Biotechnol. 37 (1) (2019) 38–44, https://doi.org/10.1038/nbt.4314.
[6] J.H. Ward, Hierarchical grouping to optimize an objective function, J. Am. Stat. Assoc. 58 (301) (1963) 236–244, https://doi.org/10.1080/01621459.1963.10500845.
[7] M. Ankerst, M.M. Breunig, H.-P. Kriegel, J. Sander, OPTICS: ordering points to identify the clustering structure, in: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, SIGMOD '99, ACM, New York, NY, USA, 1999, pp. 49–60.
[8] T. Kohonen, The self-organizing map, Proc. IEEE 78 (9) (1990) 1464–1480, https://doi.org/10.1109/5.58325.
[9] C.C. Aggarwal, A. Hinneburg, D.A. Keim, On the surprising behavior of distance metrics in high dimensional space, in: J. Van den Bussche, V. Vianu (Eds.), Database Theory — ICDT 2001: 8th International Conference London, UK, January 4–6, 2001 Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 420–434.
[10] S.K. Card, G.G. Robertson, J.D. Mackinlay, The information visualizer, an information workspace, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Reaching Through Technology - CHI '91, ACM Press, New Orleans, Louisiana, United States, 1991, pp. 181–186.
[11] A. Newell, Unified Theories of Cognition, Harvard University Press, 1994.
[12] J.-D. Fekete, R. Primet, Progressive analytics: a computation paradigm for exploratory data analysis, CoRR, arXiv:1607.05162 [abs].
[13] M. Espadoto, R.M. Martins, A. Kerren, N.S.T. Hirata, A.C. Telea, Towards a quantitative survey of dimension reduction techniques, IEEE Trans. Vis. Comput. Graph. 27 (3) (2021) 2153–2173, https://doi.org/10.1109/TVCG.2019.2944182.
[14] L.G. Nonato, M. Aupetit, Multidimensional projection for visual analytics: linking techniques with distortions, tasks, and layout enrichment, IEEE Trans. Vis. Comput. Graph. 25 (8) (2019) 2650–2673, https://doi.org/10.1109/TVCG.2018.2846735.
[15] D. Sacha, L. Zhang, M. Sedlmair, J.A. Lee, J. Peltonen, D. Weiskopf, S.C. North, D.A. Keim, Visual interaction with dimensionality reduction: a structured literature analysis, IEEE Trans. Vis. Comput. Graph. 23 (1) (2017) 241–250, https://doi.org/10.1109/TVCG.2016.2598495.
[16] B. Fritzke, A growing neural gas network learns topologies, in: G. Tesauro, D.S. Touretzky, T.K. Leen (Eds.), Advances in Neural Information Processing Systems, Vol. 7, MIT Press, 1995, pp. 625–632.
[17] M. Riveiro, F. Johansson, G. Falkman, T. Ziemke, Supporting maritime situation awareness using self organizing maps and Gaussian mixture models, in: SCAI, 2008.
[18] A. Ultsch, Kohonen's self organizing feature maps for exploratory data analysis, in: Proc. INNC90, 1990, pp. 305–308.
[19] E. Ventocilla, M. Riveiro, A comparative user study of visualization techniques for cluster analysis of multidimensional data sets, Inf. Vis. 19 (4) (2020) 318–338, https://doi.org/10.1177/1473871620922166.
[20] J.A.F. Costa, R.S. Oliveira, Cluster analysis using growing neural gas and graph partitioning, in: 2007 International Joint Conference on Neural Networks, 2007, pp. 3051–3056.
[21] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[22] E.A. Ventocilla, R.M. Martins, F.V. Paulovich, M. Riveiro, Progressive multidimensional projections: a process model based on vector quantization, in: Machine Learning Methods in Visualisation for Big Data, 2020.
[23] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113, https://doi.org/10.1145/1327452.1327492.
[24] T. Martinetz, K. Schulten, A "neural-gas" network learns topologies, in: T. Kohonen, K. Mäkisara, O. Simula, J. Kangas (Eds.), Artificial Neural Networks, Elsevier, North Holland, 1991, pp. 397–402.
[25] T. Martinetz, Competitive Hebbian learning rule forms perfectly topology preserving maps, in: S. Gielen, B. Kappen (Eds.), ICANN '93, Springer London, London, 1993, pp. 427–434.
[26] T. Mühlbacher, H. Piringer, S. Gratzl, M. Sedlmair, M. Streit, Opening the black box: strategies for increased user involvement in existing algorithm implementations, IEEE Trans. Vis. Comput. Graph. 20 (12) (2014) 1643–1652, https://doi.org/10.1109/TVCG.2014.2346578.
[27] J. Kruskal, M. Wish, Multidimensional Scaling, SAGE Publications, Inc., 2455 Teller Road, Thousand Oaks California 91320, United States of America, 1978.
[28] J.B. Tenenbaum, V. de Silva, J.C. Langford, A global geometric framework for nonlinear dimensionality reduction, Science 290 (5500) (2000) 2319–2323, https://doi.org/10.1126/science.290.5500.2319.
[29] L. McInnes, J. Healy, J. Melville, UMAP: uniform manifold approximation and projection for dimension reduction, arXiv:1802.03426.
[30] E. Ventocilla, M. Riveiro, Visual growing neural gas for exploratory data analysis, in: 10th International Conference on Information Visualization Theory and Applications, 2019, pp. 58–71.
[31] J. Fliege, W. Benn, MapReduce-based growing neural gas for scalable cluster environments, in: P. Perner (Ed.), Machine Learning and Data Mining in Pattern Recognition, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2016, pp. 545–559.
[32] A. Adam, S. Leuoth, S. Dienelt, W. Benn, Performance gain for clustering with growing neural gas using parallelization methods, in: 12th International Conference on Enterprise Information Systems, SciTePress, 2010, pp. 264–269.
[33] S. Orts, J. Garcia-Rodriguez, D. Viejo, M. Cazorla, V. Morell, GPGPU implementation of growing neural gas: application to 3D scene reconstruction, J. Parallel Distrib. Comput. 72 (10) (2012) 1361–1372, https://doi.org/10.1016/j.jpdc.2012.05.008.
[34] Y. Zhu, S. Chen, Growing neural gas with random projection method for high-dimensional data stream clustering, Soft Comput. 24 (13) (2020) 9789–9807, https://doi.org/10.1007/s00500-019-04492-4.
[35] J. Schneider, M. Vlachos, On randomly projected hierarchical clustering with guarantees, in: Proceedings of the 2014 SIAM International Conference on Data Mining, SIAM, 2014, pp. 407–415.
[36] W.N. Street, W.H. Wolberg, O.L. Mangasarian, Nuclear feature extraction for breast tumor diagnosis, in: Biomedical Image Processing and Biomedical Visualization, Vol. 1905, International Society for Optics and Photonics, 1993, pp. 861–870.
[37] W.J. Nash, T.L. Sellers, S.R. Talbot, A.J. Cawthorn, W.B. Ford, The population biology of abalone (haliotis species) in Tasmania. I. Blacklip abalone (H. rubra) from the north coast and islands of Bass Strait, Technical Report 48, Sea Fisheries Division, 1994, p. 411.
[38] D. Dua, C. Graff, UCI machine learning repository, http://archive.ics.uci.edu/ml, 2017.
[39] M. Fanty, R. Cole, Spoken letter recognition, in: Advances in Neural Information Processing Systems, 1991, pp. 220–226.
[40] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint, arXiv:1708.07747.
[41] H. Reyer, B.U. Metzler-Zebeli, N. Trakooljul, M. Oster, E. Muráni, S. Ponsuksili, F. Hadlich, K. Wimmers, Transcriptional shifts account for divergent resource allocation in feed efficient broiler chickens, Sci. Rep. 8 (1) (2018) 1–9.
[42] J. Pennington, R. Socher, C.D. Manning, Glove: global vectors for word representation, https://nlp.stanford.edu/projects/glove/. (Accessed 30 September 2020).
[43] Google code archive - long-term storage for Google code project hosting, https://code.google.com/archive/p/word2vec/. (Accessed 30 September 2020).
[44] E. Ventocilla, M. Riveiro, Visual analytics solutions as 'off-the-shelf' libraries, in: Information Visualisation (IV), 2017 21st International Conference, IEEE, 2017, pp. 281–287.
[45] E.J. Palomo, E. López-Rubio, The growing hierarchical neural gas self-organizing neural network, IEEE Trans. Neural Netw. Learn. Syst. 28 (9) (2017) 2000–2009, https://doi.org/10.1109/TNNLS.2016.2570124.