Neural Computing and Applications
https://doi.org/10.1007/s00521-020-04790-1

S.I. : EMERGING TRENDS OF APPLIED NEURAL COMPUTATION - E_TRAINCO

Evaluating graph resilience with tensor stack networks: a Keras implementation

Georgios Drakopoulos[1] · Phivos Mylonas[1]

Received: 15 November 2019 / Accepted: 17 February 2020

Abstract
In communication networks resilience or structural coherency, namely the ability to maintain total connectivity even after some data links are lost for an indefinite time, is a major design consideration. Evaluating resilience is a computationally challenging task since it often requires examining a prohibitively high number of connections or of node combinations, depending on the structural coherency definition. In order to study resilience, communication systems are treated in an abstract level as graphs where the existence of an edge depends heavily on the local connectivity properties between the two nodes. Once the graph is derived, its resilience is evaluated by a tensor stack network (TSN). TSN is an emerging deep learning classification methodology for big data which can be expressed either as stacked vectors or as matrices, such as images or oversampled data from multiple-input and multiple-output digital communication systems. As their collective name suggests, the architecture of TSNs is based on tensors, namely higher-dimensional vectors, which simulate the simultaneous training of a cluster of ordinary multilayer feedforward neural networks (FFNNs). In the TSN structure the FFNNs are also interconnected and, thus, at certain steps of the training process they learn from the errors of each other. An additional advantage of the TSN training process is that it is regularized, resulting in parsimonious classifiers. The TSNs are trained to evaluate how resilient a graph is, where the real structural strength is assessed through three established resiliency metrics, namely the Estrada index, the odd Estrada index, and the clustering coefficient. Although the approach of modelling the communication system exclusively in structural terms is function oblivious, it can be applied to virtually any type of communication network independently of the underlying technology. The classification achieved by four configurations of TSNs is evaluated through six metrics, including the F1 metric as well as the type I and type II errors, derived from the corresponding contingency tables. Moreover, the effects of sparsifying the synaptic weights resulting from the training process are explored for various thresholds. Results indicate that the proposed method achieves a very high accuracy, while it is considerably faster than the computation of each of the three resilience metrics. Concerning sparsification, after a threshold the accuracy drops, meaning that the TSNs cannot be further sparsified. Thus, their training is very efficient in that respect.

Keywords Tensor stack network · Tensor algebra · Deep learning · Big data · Higher-order data · Graph mining · Graph resilience · Estrada index · Clustering coefficient · Multilinear classification · Sparsification · Regularization · TensorFlow · Keras

Mathematics Subject Classification 05C50 · 05C62 · 05C76 · 68R10 · 94C15 · 97K30

1 Introduction

Graph resilience or graph coherency is one of the main design parameters in communication networks, since a large number of operating scenario rely heavily on total connectivity, namely the ability of any given node to reach any other node of the system, even at the face of permanent loss of a number of randomly selected data links.

✉ Georgios Drakopoulos
c16drak@ionio.gr

Phivos Mylonas
fmylonas@ionio.gr

[1] Humanistic and Social Informatics Lab, Department of Informatics, Ionian University, Corfu, Greece

Published online: 10 March 2020

Modelling in an abstract layer the communication system as a graph allows the exploration of the resilience property in a way which is independent of both the technology and the hardware used and at the same time allows researchers to concentrate only on connectivity patterns, such as local triangle density, degree distribution, and the length of the graph diameter. It should be highlighted that, although the selection of nodes is straightforward since they represent points of interest like controllers, terminals, and other communication equipment, the same is not true for edges. The latter must be very carefully picked, especially in wireless and mobile networks, by taking into account factors including effective communication ranges, realistic battery lives and energy consumption rates and by avoiding collisions like the hidden terminal problem. At any rate, within the context of this article it will be assumed that modelling is perfect.

Tensor stack network (TSN) is an emerging deep learning scheme class for big data classification as mentioned in Deng and Yu [13] and in Deng et al. [14]. The functionality of a TSN consists of simultaneously executing a cluster of ordinary feedforward neural networks (FFNNs). Each such network can learn from the errors made by the others in the cluster, effectively accelerating the training process. Stacking the matrices of synaptic weights in each TSN layer, whether input, hidden, or output, in order to be able to handle the stacked input and intermediate data results in the creation of tensors, namely vectors with three or more dimensions. Thus, the training process of a TSN essentially breaks down into a repeated sequence of tensor and matrix operations.

Tensors are ideal for expressing higher-order data with simultaneous multiple relationships, as is the case of a graph adjacency matrix, an image, or oversampled cyclostationary data from multiple-input and multiple-output (MIMO) digital communication systems as described in Ngo et al. [55] and in Larsson et al. [41]. In fact, matrices and tensors are the natural input types of a TSN in the same way vectors are for FFNNs. If an adjacency matrix were to be inserted into an FFNN, it would have first to be partitioned columnwise, rowwise, or in some other ways generating a set of training vectors. On the contrary, a matrix can be a single training element of a TSN whose higher-order architecture can process a sequence of such elements.

The primary contribution of this article is twofold. First, four different TSN configurations are created, trained with a set of synthetic Kronecker graphs, and evaluated in terms of their classification performance based on metrics derived directly from the corresponding contingency tables. Second, once the set of the best synaptic weights is determined for each configuration, it is sparsified in various degrees, meaning that synaptic weights with absolute values below a given threshold are set to zero, and the effect of this sparsification is assessed based on the same metrics. As actual resilience of each graph in both the training and the test datasets is considered a function of the Estrada index, odd Estrada index, and the clustering coefficient.

The principal motivation behind this article is that the proposed graph resilience model is both reconfigurable and higher order. Once new example graphs with certain resilience properties have been either identified in the relevant bibliography or discovered in practice from actual deployed communication networks. TSN reconfiguration, in the form of training, can be conducted efficiently from an energy-preserving perspective with only a limited number of selected representative graph samples or during times when the communication network load is low. Additionally, the higher-order nature of a TSN can take into account the effect of various structural factors such as distance distribution, network diameter, and local clustering patterns.

The remaining of this work is structured as follows. Section 2 reviews current scientific literature in the fields of tensor stack networks, neural networks, tensor algebra, and graph resilience. The proposed tensor stacked networks as well as their training process are described in Sect. 3. The experimental results and the associated analysis are the focus of Sect. 4, whereas future research directions are explored in Sect. 5. Uppercase calligraphic letters are reserved for tensors, uppercase boldface for matrices, and lowercase boldface for vectors. Finally, the notation of this article is shown in Table 1.

2 Previous work

Tensor algebra is the next evolutionary step of classical linear algebra since its focus is vectors of three or more dimensions and includes matrices and one-dimensional vectors as special yet valid cases Kolda [38]. Because of their multiplicative volume and the interdependency between their elements, tensors belong to big data Fisher [26] with many tensor operations such as contractions requiring GPUs for efficiency Shi et al. [66]. Tensors appear in many knowledge mining applications including finding higher-order digital influence in social networks as in Drakopoulos et al. [20], multilinear discriminant analysis as in Zeng et al. [77], an extension of term–document matrix to term–keyword–document third-order tensor for PubMed document retrieval based on the fact that keywords have more semantic information than ordinary terms Drakopoulos et al. [19], multifactor face recognition Vasilescu and Terzopoulos [68], and multilayer knowledge graphs which are represented by adjacency tensors Drakopoulos et al. [21]. Moreover, when the structure of complex systems is examined simultaneously under

Table 1 Notation used in this research paper

Symbol	Meaning
$\triangleq$	Definition or equality by definition
$\{s_1, \ldots, s_n\}$	Set with elements $s_1, \ldots, s_n$
$\lvert S\rvert$ or $\lvert\{s_1, \ldots, s_n\}\rvert$	Set cardinality
$\lVert\mathscr{T}\rVert_F$	Frobenius norm of tensor $\mathscr{T}$
$\otimes$	Kronecker tensor product
$\times_k$	Tensor multiplication along dimension k
vec $(\mathbf{M})$	Column stacking operator for matrix $\mathbf{M}$
deg (v)	Degree of vertex v
$(v_1, \ldots, v_n)$	Path consisting of distinct vertices $v_1, \ldots, v_n$
$((v_1, \ldots, v_n))$	Path consisting of possibly repeated vertices $v_1, \ldots, v_n$
K_n	Complete undirected graph with n vertices
B_{n_1,n_2}	Bipartite undirected graph with $n_1 + n_2$ vertices
C_n	Undirected cycle with n vertices
$Q[P]$	Indicator function for predicate P

multiple properties such as connectivity to given vertices and degree value, then the spectral clustering of the adjacency tensor can reveal partitions of these systems Benson et al. [5]. These partitions can be used to isolate anomalous segments such as nodes infected by a computer virus Bengua et al. [4] or nodes with dwindling connectivity Kumar et al. [40]. Such partitions can take various forms such as Tucker factorization Papalexakis and Faloutsos [58] or a variant of higher-order SVD Papalexakis et al. [59]. For a genetic algorithm for efficiently clustering third-order tensors containing spatial and linguistic data, see Drakopoulos et al. [23] and a higher-order tensor decomposition for large-scale clustering is proposed in Li and Boulware [44]. Tensors are treated as multidimensional or stacked memes in Blackmore [9], where some quality metrics for them are also proposed. The Kronecker tensor product has been proposed as a reliable way to quickly generate large-scale, power law graphs which densify at a nonlinear rate over time as shown in Leskovec [42], Seshadhri et al. [64], and Seshadhri et al. [65].

TSNs have been proposed among others in Deng and Yu [13] and in Deng et al. [14]. Because of their inherent ability to scale with input size, they have been applied to understanding and separating very large vocabularies as in Yu et al. [76]. Moreover, they can perform multiway classification of images with overlapping or translated objects as in Hutchinson et al. [32], information retrieval of documents where the term search space is excessively big as in Deng et al. [15], and speech recognition from huge samples with considerable voice distortions as in Deng [16]. A technique for reducing the training complexity by iteratively decoupling each input data mode is proposed in Wang et al. [72]. Although TSNs operate on huge data volumes, they are inherently protected against overfitting as shown in Grubb and Bagnell [29] where TSNs outperform FFNNs trained with the dropout method for the MNIST, pendigits, and letter datasets. Moreover, in Palangi et al. [57] TSNs achieve better signal recovery compared to ordinary FFNNs within a compressed sensing context. The increased discrimination power of stack generalization in classification is the focus of Wolpert [73]. Alternative sparse representations of TSNs are derived in Li et al. [43] for image classification. Early versions of TSNs which rely on different principles have been successfully applied to massive parallel brain simulation Pellionisz and Llinás [60]. For an extensive survey of alternative TSN architectures, see Deng [12].

Besides TSNs, a broad spectrum of machine and deep learning techniques has been implemented over GPUs in order to accelerate computations. Ordinary FFNNs for GPUs are described in Jang et al. [34] and in Oh and Jung [56], cellular neural networks for GPUs are proposed in Ho et al. [31], convex networks in Deng and Yu [13], self-organizing maps, namely unsupervised neural networks based on Hebbian learning rule, in Kohonen [37] and in Ritter [62], and a detailed spiking network implementation is described in Nageswaran [53]. This class of neural networks simulates brain activity at the biological neuron level and they form the basis of the Brian simulator, which is written entirely in Python Goodman and Brette [28]. In Sutskever et al. [67] a mechanism is described how neural networks can learn transformations between finite sequences. An overview of neural network research can be found among others in Schmidhuber [63].

Graph resilience metrics for communication systems can be cast in terms of different criteria, which can be broadly

divided to structural and functional ones. The former rely on global or local connectivity patterns, while the latter are based on the network function such as line of sight (LoS) Liberti and Rappaport [45], power transmission control Lin et al. [46], local signal power Wong and Cox [74], or channel capacity in MIMO systems as in Loyka [48] and in Biguesh and Gershman [7]. Regarding structural resilience, there are a number of options. For instance, vertex centrality and graph resilience can be computed from the resolvent of the graph adjacency matrix Estrada and Higham [25]. Any structural weaknesses can be revealed through small or relatively isolated graph communities through a regularization approach to the density of each community Kanavos et al. [36]. Metrics can be defined on routing distances Loguinov et al. [47], degree correlations Vázquez and Moreno [69], local connectivity patterns as in Najjar and Gaudiot [54] or in Ip and Wang [33], or on a weighted combination of paths and triangles given that both can be expressed in terms of the graph adjacency matrix Drakopoulos et al. [22]. Local graph density has been used as an initialization scheme for k-means Drakopoulos et al. [17]. For a comprehensive study of various approaches to graph resilience, see Alenazi and Sterbenz [3].

Deep learning software tools which can natively support and handle tensors have been recently developed. Perhaps the most popular is Google TensorFlow which is based on the dataflow graph computational paradigm with each vertex of the graph being a tensor as in Abadi [1] and in Abadi [2]. A Gaussian process generator for TensorFlow, which greatly facilitates a number of operations such as thermal noise simulation in digital communication channels, has been proposed in Matthews [51] and a visualization of TensorFlow computations in Wongsuphasawat [75]. Keras is a high-level front end for TensorFlow, allowing the easy handling of entire layers of neural networks and computational models as in Gulli and Pal [30]. Tensors are also supported in deep learning frameworks such as Theano described in Bergstra and et al [6], torch7 proposed in Collobert et al. [11], Caffe put forward in Jia [35], and the large-scale knowledge graph management system Pregel proposed in Malewicz [50]. A space-efficient data structure which can store tensors as multiway arrays is the focus of Kontopoulos and Drakopoulos [39]. A variety of tensor-based solutions for MATLAB has been developed including Tensor Toolbox, which is based on the Poblano numerical optimization toolbox described in Dunlavy et al. [24], TensorLab which places more emphasis on multilinear signal processing shown in Vervliet et al. [71], and the machine learning toolbox MatConvNet of Vedaldi and Lenc [70]. Multidimensional arrays are natively supported in many current programming languages but typically tensor operations require specialized libraries, as is the case of Breeze linear algebra suite for Scala. Additionally, the MLlib for Spark has native tensor support as stated in Meng [52].

3 Tensor stack networks

3.1 Generic architecture and training

As it can be seen from Fig. 1, TSNs have a modular architecture based on layers with identical structure which can be added or removed at will. Each such layer consists of two weight matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ and one third-order tensor $\mathscr{U}$, as shown in Fig. 2, making TSNs a natural choice for higher-order data. Specifically, each layer has the following components:

- Two matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ with the synaptic weights connecting the output of the previous layer with the two sigmoid sublayers.
- A third-order tensor $\mathscr{U}$ multilinearly combines the output of the two sigmoid sublayers and generates the

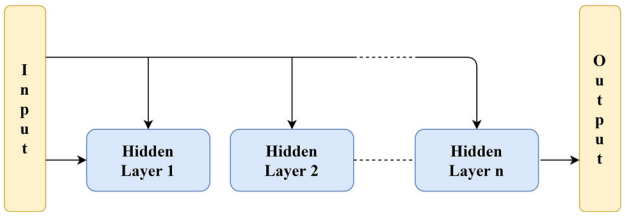

Fig. 1 General TSN architecture

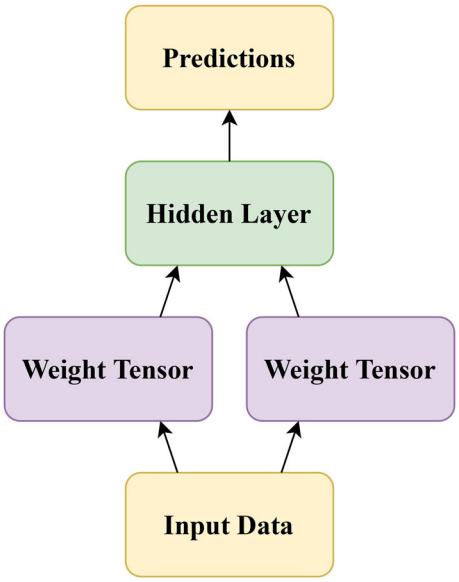

Fig. 2 Structure of a single TSN layer

predictions of that layer, which will be fed as input to the next layer.

Intuitively speaking, a p-th-order real tensor $\mathscr{T} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_p}$ is a higher-order vector indexed by p integers and belongs to $\mathbb{R}^p$. However, a tensor need not be real valued. Formally:

Definition 1 (*Tensor*) A p-th-order tensor is a linear transform connecting simultaneously p vector spaces.

Each TSN layer receives its input from the previous layer, typically codified as a matrix or a vector. Then this input is split into two parts and driven

The prediction phase for a given layer ends when the predictions matrix $\mathbf{Y}$ are generated by intermixing through tensor $\mathscr{U}$ the output of the sigmoid sublayers $\mathbf{h}_1$ and $\mathbf{h}_2$:

$$\mathbf{Y} = \mathscr{U} \times_1 (\mathbf{h}_1 \otimes \mathbf{h}_2) \tag{1}$$

By definition of the Kronecker tensor product, the resulting matrix $\mathbf{h}_1 \otimes \mathbf{h}_2$ contains each possible product of the elements of vectors $\mathbf{h}_1$ and $\mathbf{h}_2$.

Each of the vectors $\mathbf{h}_1$ and $\mathbf{h}_2$ are the output of the two parallel sigmoid sublayers operating independently as:

$$\begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{bmatrix} = \begin{bmatrix} \varphi\left(\mathbf{W}_1^T \operatorname{vec}(\mathbf{X})\right) \\ \varphi\left(\mathbf{W}_2^T \operatorname{vec}(\mathbf{X})\right) \end{bmatrix} \tag{2}$$

Notice that these intermediate outputs are independent of each other. This allows TSNs to gradually discover and learn from two different feature sets.

The input driving the sigmoid sublayers is $\operatorname{vec}(\mathbf{X})$, namely the vector resulting from stacking the columns of input matrix $\mathbf{X}$ to a very long single vector. Since the synaptic weights are fixed during the prediction phase, TSNs yield their prediction in a straightforward fashion.

During the training phase, the local error gradients with respect to $\mathscr{U}$, $\mathbf{W}_1$, and $\mathbf{W}_2$ are determined. The selected solvers applied a regularization approach in order to avoid numerical errors or overfitting similar to the one used for ill-conditioned linear systems as in Golub et al. [27] and in Bishop [8]

$$(\mathbf{A} + \mu_0 \mathbf{I})\mathbf{x} = \mathbf{b} \tag{3}$$

The regularization hyperparameter μ_0 shifts the entire spectrum of $\mathbf{A}$ away from zero at the expense of an inexact solution. An alternative to (3) would be the modified linear system:

$$(\mathbf{A} + \mathbf{D})\mathbf{x} = \mathbf{b} \tag{4}$$

In (4) the diagonal matrix $\mathbf{D}$ offers a more flexible way for regularizing the original linear system.

3.2 Implementation

The Keras package is a high-level front end for TensorFlow which allows the easy creation and manipulation of TSN layers and their initialization. In general, in order to create a deep learning model in Keras four steps are required, namely definition, translation, training, and prediction.

During the definition stage, the overall TSN architecture is defined. This includes the number of layers, whether they are sparse or not, and the connectivity between them. Each layer can be added with the model.add () method. Translation entails selecting the loss function and calling the compile.add () method to handle TSN setup. Then, it follows the compile step where the model.fit () and model.evaluate () methods are called. At the end of the TSN training pipeline comes prediction, where the model.predict () method is called and generates the actual predictions.

For each layer the nonlinear activation function was the softmax $\varphi(\cdot)$, which is often used in order to generate ranking scores of the elements of a vector $\mathbf{s}$ as:

$$\varphi(s_k) \triangleq \frac{e^{s_k}}{\sum_{j=1}^{n} e^{s_j}} \tag{5}$$

The loss function for each TSN selected was the binary negative cross-entropy K. This is a common selection for binary classification problems, since it is based on misclassification penalties instead of arbitrarily set scores. Specifically, if p is the probability mass function of the training set and q the target probability mass function, then for a training set of c samples:

$$K \triangleq \frac{1}{c} \sum_{u} \sum_{v} q(v|u) \log p(v|u) \tag{6}$$

Each layer was structurally an identical copy of the others. If $L = 6561$ and $C = 512$, then the dimensions for the input matrix $\mathbf{X}$ were $L \times L$, for matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ were $L^2 \times C$, and for $\mathscr{U}$ were $C \times L \times L$. L is the number of vertices for each of the graphs in training, testing, and validation datasets, whereas C is the number of internal features the TSN could explore in each sigmoid sublayer.

Finally, since the number of graphs in each dataset is 40, the entire dataset was fed to the TSN during each epoch.

4 Results

4.1 Overview and assumptions

The objectives of this section are:

- To describe the assumptions underlying the experimental methodology.
- To present the dataset used to derive the results.
- To explain how the various TSNs were evaluated in terms of classification.
- To explore the effects of sparsification to the TSNs, once they have been properly trained.

The underlying assumptions made in the methodology development and analysis of the results are the following. Undirected graphs are a very common model for logistics or biological networks, the primary reason being they typically based on flow or a flow-like property such as commodity supply. These properties clearly have direction which does not usually change, unless a major structural modification takes place. On the contrary, in most communication networks information usually moves along both directions between two vertices, even asymmetrically, e.g. as the case in mobile communication networks with the normal and the reverse channels.

Assumption 1 Graphs are assumed to be undirected.

Additionally, since the aim is to derive a time-efficient resilience classifier, it makes sense to assume that connectivity in a communications system remains constant during the classification procedure, assuming of course a properly trained TSN, and for a time interval sufficient for the purposes of the system. When a change takes place, then the new resilience can be computed *ab ovo* and compared with its previous values.

Assumption 2 Graphs are static.

Typically, communications networks, either local or bigger, are designed to maintain a substantial fraction of total connectivity so that each end point can communicate with another, even not necessarily directly. Bipartite graphs capture one limiting case of connectivity: The entire vertex set V is partitioned to two distinct and non-overlapping subsets V_1 and V_2. Vertices belonging to V_1 cannot directly contact each other and instead they have to relay their communications through at least one of the vertices of V_2 and vice versa. In the general case, each vertex can reach any other in a finite number of edges. This leads to the last assumption:

Assumption 3 Each graph is a single connected component.

4.2 Dataset

The training, testing, and validation datasets consist of each of 40 Kronecker graphs with 6561 vertices, 20 of which have low resilience and 20 high. In order to prevent any information leak between the three datasets, different generator graphs, each with 9 vertices, have been used. Note, it is possible to find 120 distinct generators, since in the general case with n available vertices it is possible to create an exponential number of undirected graphs in total, specifically:

$$2^{\binom{n}{2}} = 2^{\frac{n(n-1)}{2}} = 2^{\Theta(n^2)} \tag{7}$$

Recall that Kronecker graphs are generated in two steps:

- A *generator graph* with $|V|$ vertices is selected. Its properties play a crucial role in the formulation of the properties of the final Kronecker graph.
- Let $\mathbf{M} \in \{0,1\}^{|V|\times|V|}$ be the adjacency matrix of the generator graph. By repeatedly applying the Kronecker tensor product to $\mathbf{M}$ to itself, in each step results in a new adjacency matrix of a larger graph. Specifically, the adjacency matrix in step is:

$$\mathbf{M}^{[k]} \triangleq \begin{cases} \mathbf{M}, & k = 0 \\ \mathbf{M} \otimes \mathbf{M}, & k = 1 \\ \mathbf{M}^{[k-1]} \otimes \mathbf{M}, & k \geq 2 \end{cases} \tag{8}$$

The Kronecker tensor product for matrices $\mathbf{A} \in \mathbb{R}^{p\times q}$ and $\mathbf{B} \in \mathbb{R}^{u\times v}$ is defined as:

$$\mathbf{A} \otimes \mathbf{B} \triangleq \begin{bmatrix} \mathbf{A}[1,1]\mathbf{B} & \mathbf{A}[1,2]\mathbf{B} & \dots & \mathbf{A}[1,q]\mathbf{B} \\ \mathbf{A}[2,1]\mathbf{B} & \mathbf{A}[2,2]\mathbf{B} & \dots & \mathbf{A}[2,q]\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}[p,1]\mathbf{B} & \mathbf{A}[p,2]\mathbf{B} & \dots & \mathbf{A}[p,q]\mathbf{B} \end{bmatrix} \in \mathbb{R}^{pu\times qv} \tag{9}$$

Synthetic Kronecker graphs Leskovec [42] have been shown to have many of the properties of real, large-scale, graphs such as those encountered in brain connectivity networks, protein-to-protein interaction networks, transportation and logistics networks, as well as in communication systems and in the Internet graph. These properties include:

- The sorted components of the primary eigenvector of the adjacency matrix follow a power law. This suggests, in view of the eigenvector centrality, that a considerable fraction of vertices play a central role in local communication.
- The sorted eigenvalues of the adjacency matrix follow a power law. There is one large eigenvalue followed by one or two large ones and the remaining decay with a rate which is much slower than that of an exponential decay. Additionally, some tend to alternate in sign around zero as do the eigenvalues of real graphs.
- The degree distribution follows a power law. This is a very important property, as it allows easy graph scaling.

Moreover, it results in a significant fraction of vertices with a moderate degree, which facilitates local communication.
– The number of triangles is close to the observed such number of real-world, large-scale graphs. Triangles are instrumental in the local information diffusion in many scenarios.

A major advantage of synthetic graphs is that they can be chosen to contain specific patterns which influence resilience. Table 2 summarizes the properties of the graphs used in the three datasets. The upper half of this table has the low resilience graphs, whereas the lower half contains the high-resilience graphs. For each pattern exists two distinct yet isomorphic graphs which have it, so that TSNs can discover these patterns independently of row and column permutations of the adjacency matrix. All patterns exist in the three datasets. Graphs 1–20 and 61–80 have been assigned to training dataset, graphs 21–40 and 81–100 to training, and graphs 41–60 and 101–120 to validation.

Notice that the addition or deletion of an arbitrary number of edges is, according to Seshadhri et al. [65], tantamount to superimposing a power law graph with an exponential graph, eventually resulting in a new exponential graph such as those generated by the ERGM and SUGM model classes Chandrasekhar and Jackson [10]. The latter have considerably different properties than their power law counterparts and in terms of resilience are quite fragile as they typically lack the large number of local paths whose collaboration can compensate for the loss of a long-distance path Lusher et al. [49]. Thus, we are using the graphs as generated by each model.

Definition 2 (*Tensor sparsity*) The sparsity ρ_0 of a tensor $\mathscr{T} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_p}$ is defined as the ratio of the number of zero entries to the total number of entries, namely

$$\rho_0 \triangleq \frac{\sum_{i_1=1}^{I_1} \cdots \sum_{i_p=1}^{I_p} Q\left[\mathscr{T}\left[i_1, \ldots, i_p\right] = 0\right]}{\prod_{k=1}^{p} I_k} \tag{10}$$

In Eq. (10) $Q[P]$ denotes the indicator function for predicate P, namely it equals 1 if P is true and 0 otherwise. By separately testing each of the elements of $\mathscr{T}$ for equality with zero and summing the results, the total number of zero elements follows.

However intuitive it is, Eq. (10) is prone to numerical errors coming from a number of sources including finite numerical precision, limitations in floating point operations, as well as the large number of operands which may accumulate significant roundoff errors. Therefore, a tensor entry which should be zero may very well have a very

Table 2 Properties of the testing, training, and validation graphs

Indices	Low resilience pattern
1–2, 21–22, 41–42	Binary tree
3–4, 23–24, 43–44	Three K_3 with 2 connecting edges
5–6, 25–26, 45–46	$B_{2,7}$ with 10 connecting edges
7–8, 27–28, 47–48	$B_{3,6}$ with 12 connecting edges
9–10, 29–30, 49–50	$B_{4,5}$ with 14 connecting edges
11–12, 31–32, 51–52	$B_{4,5}$ with 16 connecting edges
13–14, 33–34, 53–54	Two K_4 connected through a single articulation vertex
15–16, 35–36, 55–56	Non-overlapping C_4 and C_5 connected with a single edge
17–18, 37–38, 57–58	Two non-overlapping C_4 connected through a single articulation vertex
19–20, 39–40, 59–60	K_6 with 3 vertices of degree 1
Indices	**High-resilience pattern**
61–62, 81–82, 101–102	Three overlapping C_4 with 4 edges between each cycle pair
63–64, 83–84, 103–104	Three overlapping C_5 with 6 edges between each cycle pair
65–66, 85–86, 105–106	Two K_7 with 12 connecting edges between each cycle
67–68, 87–88, 107–108	K_5 and K_4 with 8 connecting edges
69–70, 89–90, 109–110	K_5 and K_4 with 12 connecting edges
71–72, 91–92, 111–112	K_9 minus C_3
73–74, 93–94, 113–114	K_9 minus two K_3
75–76, 95–96, 115–116	Three K_3 with 8 connecting vertices
77–78, 97–98, 117–118	K_9 minus a C_4 and a C_5
78–80, 99–100, 119–120	K_9 minus a C_9

small nonzero value. In order to compensate for this, a more robust definition for sparsity is required. In this article the following definition will be used:

Definition 3 (*Effective tensor sparsity*) The effective sparsity $\rho(\tau_0)$ relative to a strictly positive threshold τ_0 of a tensor $\mathscr{R} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_p}$ is the fraction of the total number of tensor entries whose absolute value equals or is below τ_0 to the total number of entries, namely

$$\rho_0(\tau_0) \triangleq \frac{\sum_{i_1=1}^{I_1} \cdots \sum_{i_p=1}^{I_p} Q\left[\left|\mathscr{T}\left[i_1, \ldots, i_p\right]\right| \leq \tau_0\right]}{\prod_{k=1}^{p} I_k}, \quad \tau_0 > 0 \tag{11}$$

The exact computation of Eq. (11) requires $O\left(\prod_{k=1}^{p} I_k\right)$ operations to compute. Although it is linear in terms of the tensor size, as in any big data case there is the question whether something better can be done, specifically whether the order of magnitude of $\rho(\tau_0)$ for a given τ_0 can be safely estimated. Efficient set cardinality estimators based on advanced principles exist Drakopoulos et al. [18]. However, for the purposes of this article $\rho(\tau_0)$ is computed in the linear way.

Definition 4 (*Frobenius norm*) The Frobenius norm of a real p-th-order tensor denoted by $\|\mathscr{T}\|_F$ of a tensor $\mathscr{T} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_p}$ is defined as the square root of the sum of its squared elements:

$$\|\mathscr{T}\|_F \triangleq \left(\sum_{i_1=1}^{I_1} \cdots \sum_{i_p=1}^{I_p} \mathscr{T}^2\left[i_1, \ldots, i_p\right]\right)^{\frac{1}{2}} \tag{12}$$

As with tensor sparsity, the computation of the Frobenius norm following definition (12) requires linear time with respect to the total number of elements of the tensor.

4.3 Coherence evaluation

Once the dataset has been formed, the question of how the true resilience is computed. The answer lies in this case in computing three reliable and established graph resilience metrics, namely the Estrada index, the odd Estrada index, and the clustering coefficient. Each of these metrics treats graphs differently.

Definition 5 (*Estrada index*) Let $\{\lambda_k\}_{k=1}^{n}$ be the eigenvalues of a graph adjacency matrix $\mathbf{M}$ with $\lambda_1 > \lambda_2 \geq \cdots \geq \lambda_n$. Then, the Estrada index J_e which evaluates graph resilience is defined as:

$$J_e \triangleq \sum_{k=1}^{n} e^{\lambda_k} = e^{\lambda_1} + e^{\lambda_2} + \cdots + e^{\lambda_n} \tag{13}$$

Although the Estrada index is defined easily, its computation is not always easy, especially if the spectrum of $\mathbf{M}$ contains roughly equal in size subsets of very large and very small values. To avoid numerical instabilities, the Priest summation algorithm described in Priest [61] has been used. The intuition behind this algorithm is that absorption can be avoided by constantly adding numbers of comparable size. It should be noted that summation of numbers of various orders of magnitude is a concern in other computing environments. For instance, in the Julia scientific computing language the default sum function works by pairwise summation which combines high numerical accuracy, especially for standard IEEE precisions, with good performance.

An alternative to a more precise summation algorithm, which is not used in this article, would be to substitute the finite sum with a finite integral, provided that appropriate integration limits x_0 and x_1 can be found:

$$J_e \approx \int_{x_0}^{x_1} e^x dx = e^{x_1} - e^{x_0} \tag{14}$$

The odd Estrada index J_o combines two experimental observations concerning the original Estrada index, namely that adjacency matrix eigenvalues alternate in sign around zero and that only a few of the positive largest such eigenvalues actually contribute to graph resilience.

Definition 6 (*Odd Estrada index*) The odd Estrada index J_o for assessing graph resilience is defined as:

$$J_o \triangleq \sum_{k=1}^{m} \sinh \lambda_n \tag{15}$$

Notice that only $m < n$ eigenvalues are taken into account in Definition 6, since their exponentials dominate the sum. Although there is no standard formula for m, a common selection rule of thumb is:

$$m = \min\left\{\left\lceil \sqrt{n} \right\rceil, p\right\} \tag{16}$$

In Eq. (16) p is the number of strictly positive eigenvalues.

The rationale behind the odd Estrada index is that only paths of odd length should be considered, since paths of even length are likely to consist of smaller repeated paths. For instance, a path of length four can very well have the form:

$$\left(\left(v_i, v_j, v_i, v_j\right)\right) \tag{17}$$

This is equivalent to count twice the contribution of edge (v_i, v_j), which has been already accounted for. Clearly, such paths should not count towards graph resilience. One way of ensuring that only odd paths are considered is to apply an odd function to the spectrum of the adjacency matrix. Hyperbolic sinus is a numerically stable and odd function, since its Taylor expansion is:

$$\sinh x = \sum_k \frac{x^k}{k!}, \qquad k \equiv 1 (\mathrm{mod}\ 2) \tag{18}$$

Clustering coefficient is an alternative graph structural coherence metric which relies on the fact that triangles, the smallest graph communities, play an instrumental role in the efficient local information diffusion and in the compensation of communication caused by lost long haul edges with a path of local edges. Consequently, the more triangles exist, the more coherent a graph is.

Definition 7 (*Clustering coefficient*) The clustering coefficient J_c is defined as the arithmetic mean of the ratio of the number of triangles a vertex v_k actually is a member of to the maximum number of triangles v_k can participate to.

$$J_c \triangleq \frac{1}{|V|} \sum_{v_k \in V} \frac{\mathbf{M}^3[k,k]}{\binom{\deg(v_k)}{3}} \approx \frac{1}{|V|} \sum_{v_k \in V} \frac{\mathbf{M}^3[k,k]}{\deg(v_k)^3} \tag{19}$$

The numerator in (19) is the k-th diagonal element of $\mathbf{M}^3$, where $\mathbf{M}$ is the graph adjacency matrix. The approximation formula is derived from the fact that for sufficiently large n and small k it holds that $\binom{n}{k} \approx n^k$.

Figure 3 shows the sum of the logarithms of the three resilience scores for each of the 120 graphs of the dataset, namely:

$$J = \log J_e + \log J_o + \log J_c \tag{20}$$

The logarithms were used in order to use scores comparable in scale. It is clear that the two clusters of low- and high-resilience graphs are distinguishable.

In order to understand the role of the spectrum of the adjacency matrix, the spectra of the graph with the lowest and the highest score J are shown in Fig. 4. From the plot, it becomes clear that both spectra have a small number of large positive eigenvalues followed successively by a large number of medium and low positive eigenvalues, a very long string of zero or almost zero eigenvalues, and by a few large negative eigenvalues. This pattern appears in both spectra and justifies the use of only a few large eigenvalues in computing both the original and the odd Estrada indices.

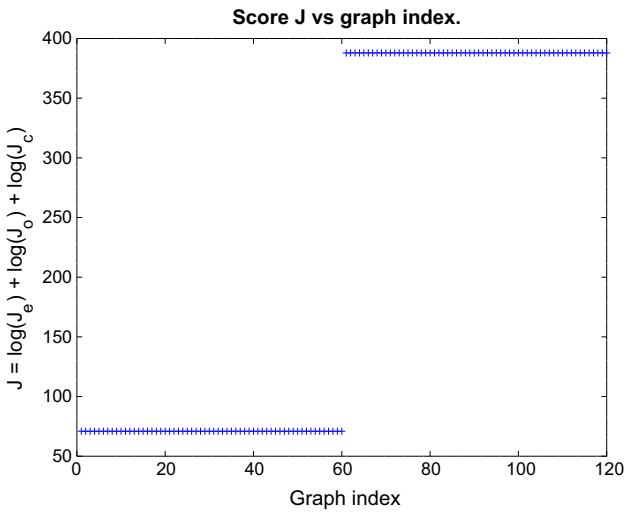

Fig. 3 Score J for the entire dataset

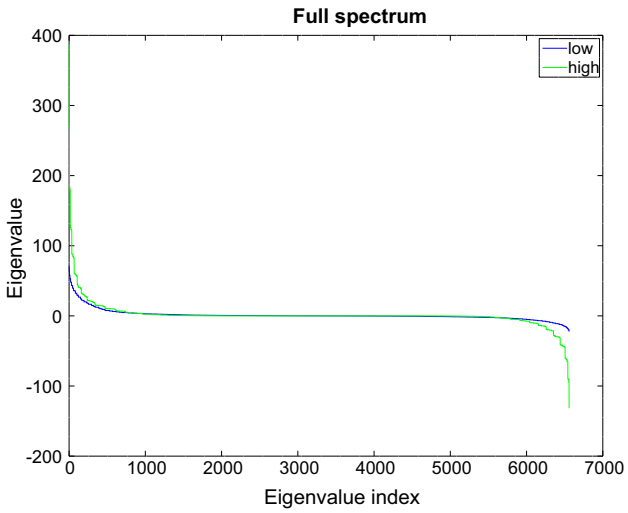

Fig. 4 Spectrum of the adjacency matrices with the lowest and the highest J score

It is worth mentioning that there is a variance between the two graph categories; however, it is not easily discernible in Fig. 3. This happens because different graph patterns lead to different adjacency matrices. The latter in the general case have different spectra and, hence, different Estrada and odd Estrada indices. Additionally, different connectivity patterns lead to distinct clustering coefficients.

The m largest positive eigenvalues as determined by Eq. (16) for the above graphs are shown in Fig. 5 in logarithmic scale. From the plot the difference in the order of magnitude is evident. Additionally, the eigenvalues appear to be highly clustered, which holds up to an extent in large, real world graphs and in the particular case is a result of the Kronecker product.

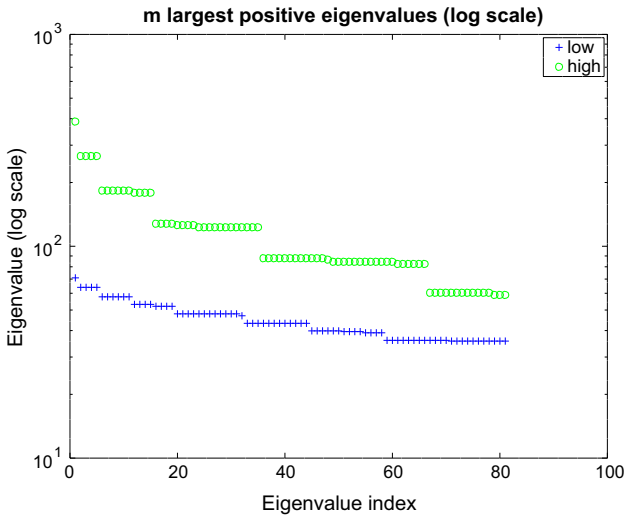

Fig. 5 The m largest positive eigenvalues of the above (log scale)

4.4 Architectures

Two of the most important parameters of TSNs are the number of layers and the way the synaptic weights are initialized. Depending on the error criterion used the TSN, the synaptic weights may theoretically converge to a single global minimum in the weight space or such a minimum may not be guaranteed. In practice, the synaptic weights may be trapped to a local minimum as a result of the stochastic nature of initialization process.

In order to keep training complexity and total response low, two fully connected architectures were tested. The first has two hidden layers, while the second has one. These architectures are shown, respectively, in Figs. 6 and 7. Moreover, the role of each component in its respective architecture is explained in Tables 3 and 4. An important feature of both architectures, inherited from the generic TSN structure, is that input adjacency matrices need not be vectorized but they maintain their square structure. This crucially differentiates TSNs from ordinary FFNNs and other neural network architectures whose input is strictly one-dimensional vectors. Also, this implies that the TSN structure is adjusted each time to the dimensions of the original adjacency matrix.

The primary difference of the architectures is the additional hidden layer in the first architecture. Otherwise, the synaptic weights of both architectures are trained exactly in the same way, notwithstanding the different parameters and local error gradient matrices, especially those used in the additional hidden layer of the first architecture. Notice that the second architecture is rather a special case, as it has only one hidden layer. This layer makes it similar to an extreme learning machine (ELM), although the latter has a single layer populated with a huge number of neurons. Notice that in Fig. 6 the two hidden layers are shown in different colours.

Additionally, two synaptic weight initialization processes were used for each architecture. The first is based on a standard Gaussian distribution and the second on a uniform distribution on $[-1, +1]$. As mentioned earlier, the cost function was the negative binary cross-entropy. In each epoch all the test matrices were projected to the TSN.

Table 5 summarizes the contingency tables for each of the four cases. Notice that it was created using the test set

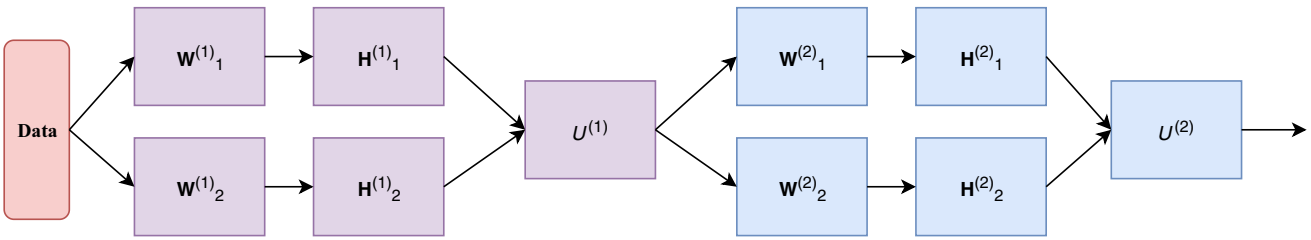

Fig. 6 The first TSN architecture

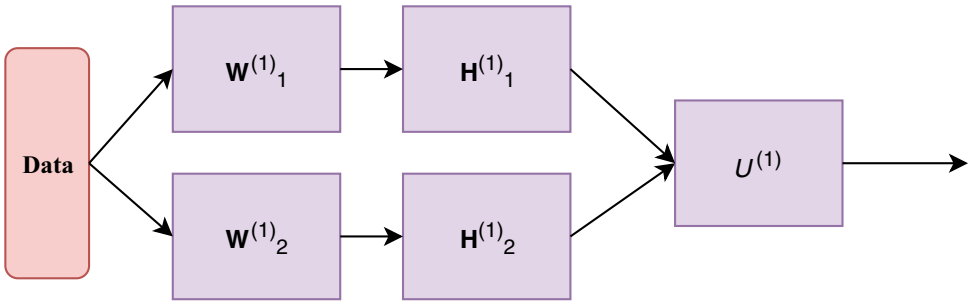

Fig. 7 The second TSN architecture

Table 3 Components of the first TSN architecture

Component	Role
Data	Input graph represented as an $n \times n$ adjacency matrix
$\mathbf{W}_1^{(1)}$	Synaptic weights connecting the data layer with $\mathbf{H}_1^{(1)}$
$\mathbf{W}_2^{(1)}$	Synaptic weights connecting the data layer with $\mathbf{H}_2^{(1)}$
$\mathbf{H}_1^{(1)}$	First neuron stack of the first hidden layer
$\mathbf{H}_2^{(1)}$	Second neuron stack of the first hidden layer
$\mathscr{U}^{(1)}$	Tensor combining the outputs of the first hidden layer neuron stacks
$\mathbf{W}_1^{(2)}$	Synaptic weights connecting the first hidden layer with $\mathbf{H}_1^{(2)}$
$\mathbf{W}_2^{(2)}$	Synaptic weights connecting the first hidden layer with $\mathbf{H}_2^{(2)}$
$\mathbf{H}_1^{(2)}$	Upper neurons of the second hidden layer
$\mathbf{H}_2^{(2)}$	Upper neurons of the second hidden layer
$\mathscr{U}^{(2)}$	Tensor combining the outputs of the first hidden layer neuron stacks

Table 4 Components of the second TSN architecture

Component	Role
Data	Input graph represented as an $n \times n$ adjacency matrix
$\mathbf{W}_1^{(1)}$	Synaptic weights connecting the data layer with $\mathbf{H}_1^{(1)}$
$\mathbf{W}_2^{(1)}$	Synaptic weights connecting the data layer with $\mathbf{H}_2^{(1)}$
$\mathbf{H}_1^{(1)}$	First neuron stack of the single hidden layer
$\mathbf{H}_2^{(1)}$	Second neuron stack of the single hidden layer
$\mathscr{U}^{(1)}$	Tensor combining the outputs of the hidden layer neuron stacks

Table 5 Contingency tables for the four architectures after training

fa/fo	True	False	fa/so	True	False	sa/fo	True	False	sa/so	True	False
pos	20	0	pos	20	0	pos	18	2	pos	17	3
neg	19	1	neg	17	3	neg	18	2	neg	16	4

once training was complete, whereas the validation set was not utilized in these experiments.

Once the contingency table for each architecture and each synaptic weight initialization option was formed, the following evaluation metrics were computed:

- Accuracy (**ac**).
- Type I error (**t1**).
- Type II error (**t2**).
- Precision (**pr**).
- Recall (**rc**).
- F1 score (**f1**).

Table 6 summarizes the above metrics for the four TSN configurations, the latter being expressed as a combination of architecture and weight initialization option. From this table, it can be inferred that two layers are preferable to only one in terms of accuracy, precision, Type I error, and F1 score. This can be attributed to the greater flexibility offered by two hidden layers, which can represent more nonlinear functions with increased degrees of freedom. This allows for correctly separating more matrix sets. Another observation is that initializing the weights according to a Gaussian seems to yield better results. This can be explained from the fact that the Gaussian distribution has the maximum differential entropy among the class of distributions with the same variance, meaning that its values cover as much the weight space as possible. Thus, it is more probable that the initialized weights are closer to the actual weights minimizing the TSN cost function.

After training is complete, the validation dataset was fed to each of the four TSNs once and without any synaptic weight update, preventing thus any information leak from the dataset to the networks. Validation is a way of independently evaluating how well each TSN performs. In this

Table 6 Metrics for the four TSN configurations after training

Conf	ac	t1	t2	pr	rc	f1
fa/fo	0.9750	0.0000	0.0476	1.0000	0.9524	0.9756
fa/so	0.9250	0.0000	0.1304	1.0000	0.8696	0.9302
sa/fo	0.9000	0.1000	0.1000	0.9000	0.9000	0.9000
sa/so	0.8250	0.1579	0.1905	0.8500	0.8095	0.8293

Table 7 Contingency tables for the four architectures after validation

fa/fo	True	False	fa/so	True	False	sa/fo	True	False	sa/so	True	False
pos	20	0	pos	19	1	pos	18	2	pos	17	3
neg	19	1	neg	17	3	neg	18	2	neg	17	3

Table 8 Metrics for the four TSN configurations after validation

Conf	ac	t1	t2	pr	rc	f1
fa/fo	0.9750	0.0000	0.0476	1.0000	0.9524	0.9756
fa/so	0.9000	0.0555	0.1363	0.9500	0.8636	0.9047
sa/fo	0.9000	0.1000	0.1000	0.9000	0.9000	0.9000
sa/so	0.8500	0.1500	0.1505	0.8500	0.8500	0.8500

way, four new contingency tables were formed based on the results of the validation dataset which are summarized in Table 7.

Along a similar line of reasoning, the same six metrics are computed from Table 7 and their values are shown in Table 8. From the latter it can be seen that the first architecture/first option configuration and the second architecture/first options achieved the same results, whereas the other two configurations had minor differences in performance, either positive or negative. This is an indication that TSNs can successfully generalize in unknown datasets, at least in these datasets which contain familiar connectivity patterns.

As stated earlier, the total wallclock time will also play the role of an important benchmark, the reason being that TSNs, once trained, should be considerably quicker than the computation of the actual resilience metrics. In Table 9 can be seen statistics in seconds regarding TSN training, testing, and validation, and the computation of the three resilience metrics. Since the above are executed in GPU, thus freeing CPU, and no other processes were running at the time, the wallclock time is a rather accurate approximation of the total execution time.

The most costly operation for J_e and J_o is the computation of the entire spectrum of the adjacency matrix, even though only a few eigenvalues play a central role, and for J_c the cubing of the adjacency matrix. Also, it is observed that the time for training, testing, and validation is proportional to the TSN size in terms of hidden layers, whereas the three coherency metrics have constant times, since they have exact computations with closed-form summations.

4.5 Sparsification evaluation

Each deep learning model must be parsimonious in order to be interpretable and understandable. In the case of sparsification, it is clear that a suitable threshold τ_0 is the key for revealing additional zero patterns and assessing tensor robustness. This raises the question of how such a threshold is selected. Within the context of TSNs, there are a number of answers. One approach is to select τ_0 according to the order of magnitude of the root mean square value $\bar{\tau}$ defined as:

$$\bar{\tau} \triangleq \left(\frac{\|T\|_F^2}{\prod_{k=1}^{p} I_k} \right)^{\frac{1}{2}} = \left(\left(\prod_{k=1}^{p} I_k \right)^{-1} \sum_{i_1=1}^{I_1} \cdots \sum_{i_p=1}^{I_p} \mathscr{T}^2 \left[i_1, \ldots, i_p \right] \right)^{\frac{1}{2}} \tag{21}$$

The order of magnitude for each of the four TSNs is shown in Table 10.

In order to understand how robust are TSNs as classifiers, the effects of sparsifying the trained TSNs will be evaluated. First, it will be examined the effective sparsity for various values of the threshold τ_0:

$$\left| \frac{\rho_0(\tau_0) - \rho_0}{\rho_0} \right|, \quad \rho_0 \neq 0 \tag{22}$$

For the first architecture which has two hidden layers, the average sparsity of synaptic weights for tensors $\mathscr{U}_1$ and $\mathscr{U}_2$ is taken into account in order to derive a single metric. Of course, for the second architecture only the tensor $\mathscr{U}$ of the single hidden layer is considered. Figure 8 shows the sparsity for different threshold values.

By examining Fig. 8, the second architecture has the quickest growth rate of the relative increase, meaning that there are many synaptic weights which have low value. Since each weight that is zeroed out contributes equally,

Table 9 Time statistics (wallclock time, sec)

	Train	Test	Validate	J_e	J_o	J_c
fa/fo						
Min	213.4517	23.1123	23.1209	42.1259	42.2117	212.9973
Avg	235.6696	24.1426	24.2091	43.1266	43.4609	213.2517
Max	237.6183	25.6602	25.8896	45.3281	46.0031	214.5515
Std	6.3312	1.0953	1.06512	1.0324	1.0901	7.0892
fa/so						
Min	212.4517	22.8926	23.5093	42.2891	42.6572	212.6201
Avg	235.5112	23.4915	23.3217	43.6792	43.5542	214.7767
Max	237.3169	24.7991	24.4084	44.8993	45.2316	215.6653
Std	6.2996	1.0032	1.01437	1.1053	1.1188	7.1162
sa/fo						
Min	133.5549	13.8085	14.1254	42.1167	42.1992	212.0022
Avg	136.6513	14.3376	15.2281	42.9912	43.3999	213.1189
Max	138.3314	15.7176	15.9896	43.8013	44.1097	214.4477
Std	5.1121	1.1199	1.2121	1.5512	1.1716	7.0999
sa/so						
Min	133.0040	13.1688	14.2305	42.2823	42.0001	213.3334
Avg	136.2111	14.5721	15.4523	44.2212	44.0012	214.6672
Max	137.4480	15.6099	16.3331	45.5678	45.6724	215.8816
Std	5.3779	1.1094	1.1444	1.0974	1.0901	7.1222

Table 10 Root mean square order of magnitude $\bar{\tau}$

fa/fo	fa/so	sa/fo	sa/so
$10^{0.34}$	$10^{0.35}$	$10^{0.34}$	$10^{0.34}$

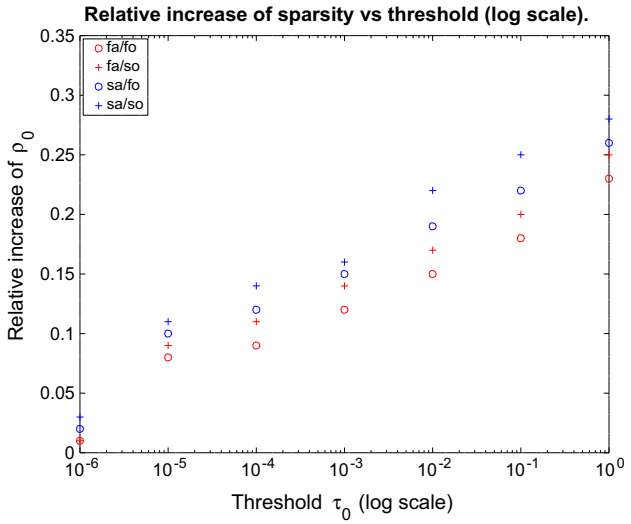

Fig. 8 Relative increase in sparsity

the relative increase in sparsity can be seen even for low threshold values.

Since elements of the tensors are zeroed out, $\|\mathscr{U}_1\|_F + \|\mathscr{U}_2\|_F$ for the first architecture or $\|\mathscr{U}_1\|_F$ for the second

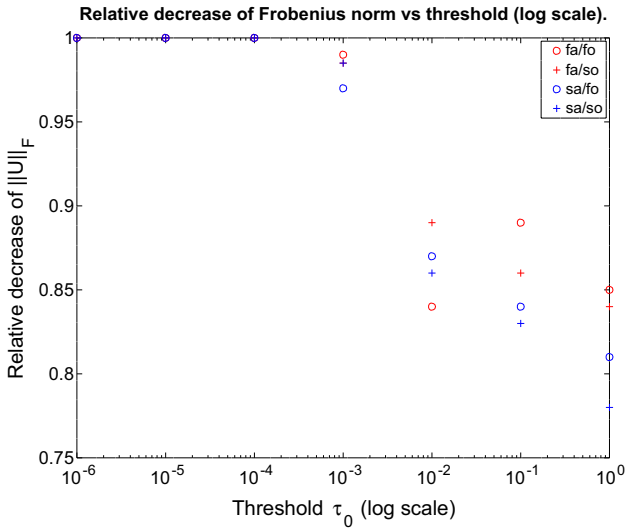

Fig. 9 Relative decrease in Frobenius norm

architecture should be decreasing as τ_0 increases. In Fig. 9 the relative decrease in these quantities is shown. For small values of τ_0, there is almost no change. However, when τ_0 reaches 10^{-2}, the sparsification effect starts becoming discernible. Again, the first architecture appears to be more robust than the second one.

Finally, the accuracy of each sparsified TSN as a percentage of the accuracy of the same architecture is shown in Fig. 10. Once the final synaptic weights have been finalized at the end of the training process, the validation

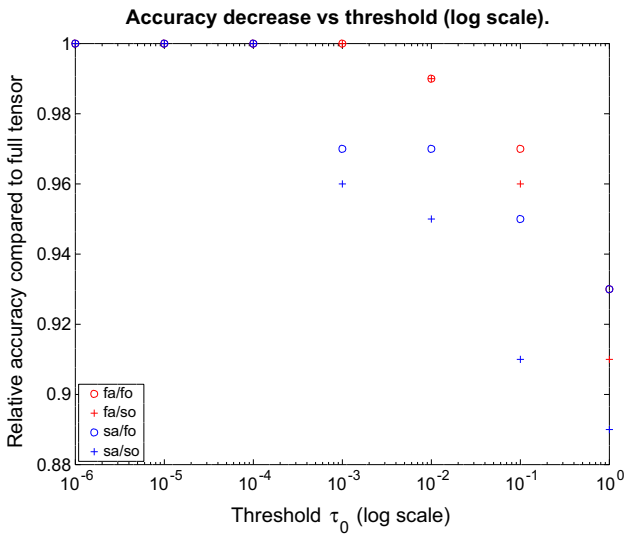

Fig. 10 Accuracy as a percentage of the initial accuracy

dataset was used to compute the accuracy. From this figure, it follows that the first architecture displays considerably high accuracy which drops only when τ_0 exceeds 10^{-2}.

Combining the findings of Figs. 8, 9, and 10, it is clear that the first architecture is a more robust classifier compared to the second. However, even the latter maintains very high accuracy even at the face of multiple sparsifications.

5 Conclusions

Structural resilience, namely the ability to maintain its connectivity at the face of loss of edges or even vertices, is an important functional specification of current communication systems, especially for wireless and mobile ones. Therefore, having a quick and reliable method for evaluating resilience is a valuable tool for researchers and field engineers alike. This article describes in detail a tensor stack network (TSN), an emerging deep learning technique for combining ordinary feedforward neural networks (FFNNs) into a cluster, training them simultaneously, and cross-training them up to an extent during distinct phases of the training process so that each FFNN can learn from the errors of other such networks. The objective is to make TSNs efficient and reliable graph resilience classifiers. Because of the inherently higher-order architecture of TSNs, they are a natural choice for using graph adjacency matrices as their basic training element. Four different configurations, deriving from two architectures and two policies for synaptic weight initialization, are trained and tested with two sets of synthetic Kronecker graphs. The real resilience of the graphs is a function of three established metrics, namely the Estrada index, the odd Estrada index, and the clustering coefficient. Moreover, once the final set of synaptic weights has been determined, a sparsification process was applied in order to determine how robust TSNs are. Results indicate a very high level of both classification accuracy and robustness.

Selecting the appropriate architecture for a given non-trivial problem is more an art than a science and entails considerable fine-tuning given the huge number of parameters involved and the interactions between them. Metrics combining structural and synaptic weight sparsity constraints should be developed, perhaps using as basis principles like AIC, BIC, or MDL. Also, the location of zero weights may reveal interesting patterns. Tensors arising from the discretization of differential equations of domain decomposition methods have such distinct patterns. In communications networks systematic zero locations may reveal lack of connectivity in some region for a number of reasons depending on the underlying technology such as excessive electromagnetic interference or mountains in the case of radio networks, strong underwater currents in the case of underwater acoustic systems, or lack of LoS in the case of microwave communications.

Concerning classification, the proposed method can be compared with TSNs or other deep learning methods for image classification, since the graph adjacency matrix can be treated as a bitmap image. Also, this article focused exclusively on graph structure and did not model any network function. Structural methods have the advantage of being function oblivious and, thus, they can be applied to literally any network. However, the objective of setting up a network in the first place is to perform a set of functions. Therefore, a model which takes into account both structure and functionality would have additional descriptive power. If functionality can be represented with structural terms, as it has been shown to be feasible in some cases in social network analysis, then extending TSNs like the ones used in this work can yield resilience classifiers.

Acknowledgements The authors acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

Compliance with ethical standards

Conflicts of interest The authors declare that they have no conflict of interest.

References

1. Abadi M (2016a) TensorFlow: learning functions at scale. ACM SIGPLAN Not. 51(9):1–1
2. Mea A (2016b) TensorFlow: a system for large-scale machine learning. OSDI 16:265–283

3. Alenazi MJ, Sterbenz JP (2015) Comprehensive comparison and accuracy of graph metrics in predicting network resilience. In: DRCN, IEEE, pp 157–164
4. Bengua JA, Phien HN, Tuan HD (2015) Optimal feature extraction and classification of tensors via matrix product state decomposition. In: ICBD, IEEE, pp 669–672
5. Benson AR, Gleich DF, Leskovec J (2015) Tensor spectral clustering for partitioning higher-order network structures. In: ICDM, SIAM, pp 118–126
6. Bergstra J et al (2011) Theano: Deep learning on GPUs with Python. In: NIPS BigLearning workshop vol 3, pp 1–48
7. Biguesh M, Gershman AB (2006) Training-based MIMO channel estimation: a study of estimator tradeoffs and optimal training signals. IEEE Trans Signal Process 54(3):884–893
8. Bishop CM (1995) Training with noise is equivalent to Tikhonov regularization. Neural Comput 7(1):108–116
9. Blackmore S (2000) The meme machine. Oxford Universtiy Press, Oxford
10. Chandrasekhar AG, Jackson MO (2014) Tractable and consistent random graph models. Technical report, National Bureau of Economic Research
11. Collobert R, Kavukcuoglu K, Farabet C (2011) torch7: A MATLAB-like environment for machine learning. In: BigLearn, NIPS workshop
12. Deng L (2014) A tutorial survey of architectures, algorithms, and applications for deep learning. APSIPA Trans Signal Inf Process 3:2 3https://doi.org/10.1017/atsip.2013.9
13. Deng L, Yu D (2011) Deep convex net: A scalable architecture for speech pattern classification. In: Twelfth annual conference of the International Speech Communication Association
14. Deng L, Hutchinson B, Yu D (2012) Parallel training for deep stacking networks. In: Thirteenth annual conference of the International Speech Communication Association
15. Deng L, He X, Gao J (2013) Deep stacking networks for information retrieval. In: ICASSP, IEEE
16. Deng L (2013) Recent advances in deep learning for speech research at Microsoft. In: ICASSP, IEEE
17. Drakopoulos G, Gourgaris P, Kanavos A, Makris C (2016a) A fuzzy graph framework for initializing k-means. IJAIT 25(6):1–21
18. Drakopoulos G, Kontopoulos S, Makris C (2016) Eventually consistent cardinality estimation with applications in biodata mining. In: SAC, ACM
19. Drakopoulos G, Kanavos A, Karydis I, Sioutas S, Vrahatis AG (2017) Tensor-based semantically-aware topic clustering of biomedical documents. Computation 5(3):34
20. Drakopoulos G, Kanavos A, Mylonas P, Sioutas S (2017) Defining and evaluating Twitter influence metrics: a higher order approach in Neo4j. SNAM 71(1):52
21. Drakopoulos G, Kanavos A, Tsolis D, Mylonas P, Sioutas S (2017) Towards a framework for tensor ontologies over Neo4j: representations and operations. In: IISA
22. Drakopoulos G, Liapakis X, Tzimas G, Mylonas P (2018) A graph resilience metric based on paths: higher order analytics with GPU. In: ICTAI, IEEE
23. Drakopoulos G, Stathopoulou F, Kanavos A, Paraskevas M, Tzimas G, Mylonas P, Iliadis L (2019) A genetic algorithm for spatiosocial tensor clustering: exploiting TensorFlow potential. Evol Syst
24. Dunlavy DM, Kolda TG, Acar E (2010) Poblano v1. 0: A MATLAB toolbox for gradient-based optimization
25. Estrada E, Higham DJ (2010) Network properties revealed through matrix functions. SIAM Rev 52(4):696–714
26. Fisher DH (1987) Knowledge acquisition via incremental conceptual clustering. Mach Learn 2(2):139–172
27. Golub GH, Hansen PC, O'Leary DP (1999) Tikhonov regularization and total least squares. J Matrix Anal Appl 21(1):185–194
28. Goodman DF, Brette R (2009) The brian simulator. Front Neurosci 3(2):192
29. Grubb A, Bagnell JA (2013) Stacked training for overfitting avoidance in deep networks. In: ICML workshops, p 1
30. Gulli A, Pal S (2017) Deep learning with keras. PACKT Publishing Ltd, Birmingham
31. Ho TY, Lam PM, Leung CS (2008) Parallelization of cellular neural networks on GPU. Pattern Recognit 41(8):2684–2692
32. Hutchinson B, Deng L, Yu D (2013) Tensor deep stacking networks. TPAMI 35(8):1944–1957
33. Ip WH, Wang D (2011) Resilience and friability of transportation networks: evaluation, analysis and optimization. IEEE Syst J 5(2):189–198
34. Jang H, Park A, Jung K (2008) Neural network implementation using CUDA and OpenMP. In: DICTA'08, IEEE, pp 155–161
35. Jia Y (2014) Caffe: convolutional architecture for fast feature embedding. In: International conference on multimedia. ACM, pp 675–678
36. Kanavos A, Drakopoulos G, Tsakalidis A (2017) Graph community discovery algorithms in Neo4j with a regularization-based evaluation metric. In: WEBIST
37. Kohonen T (1998) The self-organizing map. Neurocomputing 21(1):1–6
38. Kolda T (2009) Tensor decompositions and applications. SIAM Rev 51(3):455–500
39. Kontopoulos S, Drakopoulos G (2014) A space efficient scheme for graph representation. In: ICTAI, IEEE
40. Kumar R, Sahni A, Marwah D (2015) Real time big data analytics dependence on network monitoring solutions using tensor networks and its decompositions. Netw Complex Syst 5(2)
41. Larsson EG et al (2014) Massive MIMO for next generation wireless systems. IEEE Commun Mag 52(2):186–195
42. Jea L (2010) Kronecker graphs: an approach to modeling networks. JMLR 11:985–1042
43. Li J, Chang H, Yang J (2015) Sparse deep stacking network for image classification. In: AAAI, pp 3804–3810
44. Li L, Boulware D (2015) High-order tensor decomposition for large-scale data analysis. In: ICBD, IEEE, pp 665–668
45. Liberti JC, Rappaport TS (1996) A geometrically based model for line-of-sight multipath radio channels. Veh Technol Conf 2:844–848
46. Lin S et al (2016) ATPC: adaptive transmission power control for wireless sensor networks. TOSN 12(1):6
47. Loguinov D, Casas J, Wang X (2005) Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. IEEE/ACM TON 13(5):1107–1120
48. Loyka SL (2001) Channel capacity of MIMO architecture using the exponential correlation matrix. IEEE Commun Lett 5(9):369–371
49. Lusher D, Koskinen J, Robins G (2013) Exponential random graph models for social networks: theory, methods, and applications. Cambridge University Press, Cambridge
50. Malewicz G (2010) Pregel: a system for large-scale graph processing. In: CIKM, ACM, pp 135–146
51. Matthews DG (2017) GPflow: a Gaussian process library using tensorflow. JMLR 18(1):1299–1304
52. Xea M (2016) MLlib: machine learning in Apache spark. JMLR 17(1):1235–1241
53. Nageswaran JM (2009) A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. Neural Netw 22(5):791–800
54. Najjar W, Gaudiot JL (1990) Network resilience: a measure of network fault tolerance. ToC 2(1):174–181

55. Ngo HQ, Larsson EG, Marzetta TL (2013) Energy and spectral efficiency of very large multiuser MIMO systems. ToC 61(4):1436–1449
56. Oh KS, Jung K (2004) GPU implementation of neural networks. Pattern Recognit 37(6):1311–1314
57. Palangi H, Ward RK, Deng L (2013) Using deep stacking network to improve structured compressed sensing with multiple measurement vectors. In: ICASSP, pp 3337–3341
58. Papalexakis EE, Faloutsos C (2015) Fast efficient and scalable core consistency diagnostic for the PARAFAC decomposition for big sparse tensors. In: ICASSP, pp 5441–5445
59. Papalexakis EE, Pelechrinis K, Faloutsos C (2014) Spotting misbehaviors in location-based social networks using tensors. In: WWW, pp 551–552
60. Pellionisz A, Llinás R (1979) Brain modeling by tensor network theory and computer simulation. The cerebellum: Distributed processor for predictive coordination. Neuroscience 4(3):323–348
61. Priest DM (1991) Algorithms for arbitrary precision floating point arithmetic. In: Tenth symposium on computer arithmetic. IEEE, pp 132–143
62. Hea R (1992) Neural computation and self-organizing maps: an introduction. Addison-Wesley Reading, Boston
63. Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117
64. Seshadhri C, Pinar A, Kolda TG (2011) An in-depth study of stochastic Kronecker graphs. In: ICDM, SIAM, pp 587–596
65. Seshadhri C, Pinar A, Kolda TG (2013) An in-depth analysis of stochastic Kronecker graphs. JACM 60(2):13
66. Shi Y, Niranjan U, Anandkumar A, Cecka C (2016) Tensor contractions with extended BLAS kernels on CPU and GPU. In: HiPC, IEEE, pp 193–202
67. Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: NIPS, pp 3104–3112
68. Vasilescu MAO, Terzopoulos D (2002) Multilinear analysis of image ensembles: Tensorfaces. In: European conference on computer vision. Springer, pp 447–460
69. Vázquez A, Moreno Y (2003) Resilience to damage of graphs with degree correlations. Phys Rev E 67(1):15–101
70. Vedaldi A, Lenc K (2015) Matconvnet: Convolutional neural networks for MATLAB. In: International conference on multimedia. ACM, pp 689–692
71. Vervliet N, Debals O, De Lathauwer L (2016) TensorLab 3.0—numerical optimization strategies for large-scale constrained and coupled matrix-tensor factorization. In: Asilomar conference on signals, systems and computers. IEEE, pp 1733–1738
72. Wang M et al (2018) Disentangling the modes of variation in unlabelled data. TPAMI 40(11):2682–2695
73. Wolpert DH (1992) Stacked generalization. Neural Netw 5(2):241–259
74. Wong D, Cox DC (1999) Estimating local mean signal power level in a Rayleigh fading environment. TVT 48(3):956–959
75. Wongsuphasawat K (2018) Visualizing dataflow graphs of deep learning models in TensorFlow. Trans Vis Comput Graph 24(1):1–12
76. Yu D, Deng L, Seide F (2013) The deep tensor neural network with applications to large vocabulary speech recognition. Trans Audio Speech Language Process 21(2):388–396
77. Zeng R, Wu J, Senhadji L, Shu H (2015) Tensor object classification via multilinear discriminant analysis network. In: ICASSP, IEEE, pp 1971–1975