Neural Computing and Applications
https://doi.org/10.1007/s00521-021-06235-9

SPECIAL ISSUE ON ADVANCES OF NEURAL COMPUTING PHASING CHALLENGES IN THE ERA OF 4TH INDUSTRIAL REVOLUTION

Transform-based graph topology similarity metrics

Georgios Drakopoulos[1] · Eleanna Kafeza[2] · Phivos Mylonas[1] · Lazaros Iliadis[3]

Received: 3 March 2021 / Accepted: 13 June 2021

Abstract
Graph signal processing has recently emerged as a field with applications across a broad spectrum of fields including brain connectivity networks, logistics and supply chains, social media, computational aesthetics, and transportation networks. In this paradigm, signal processing methodologies are applied to the adjacency matrix, seen as a two-dimensional signal. Fundamental operations of this type include graph sampling, the graph Laplace transform, and graph spectrum estimation. In this context, topology similarity metrics allow meaningful and efficient comparisons between pairs of graphs or along evolving graph sequences. In turn, such metrics can be the algorithmic cornerstone of graph clustering schemes. Major advantages of relying on existing signal processing kernels include parallelism, scalability, and numerical stability. This work presents a scheme for training a tensor stack network to estimate the topological correlation coefficient between two graph adjacency matrices compressed with the two-dimensional discrete cosine transform, augmenting thus the indirect decompression with knowledge stored in the network. The results from three benchmark graph sequences are encouraging in terms of mean square error and complexity especially for graph sequences. An additional key point is the independence of the proposed method from the underlying domain semantics. This is primarily achieved by focusing on higher-order structural graph patterns.

Keywords Graph signal processing · Graph topology · Signal processing kernels · Discrete cosine transform · Tensor algebra · Tensor stack network · Julia

Mathematics subject classification 42A16 · 46E20 · 62H30 · 62P25 · 91C20 · 91C99 · 91D10 · 91D30 · 91D99

1 Introduction

Graph signal processing (GSP) over the past few years has yielded many advanced analytics for graph mining applications including topological similarity. One major metric expressing the latter is correlation coefficient between two graphs, either deterministic or stochastic. This can be the building block of graph distance metrics or clustering schemes. Given that graph similarity analytics frequently face scalability issues not only in terms of vertices and edges, but also of higher-order patterns to be discovered, it makes perfect sense for efficient computational kernels to be used.

Deep learning (DL) yields scalable and robust models which can inherently extract non-trivial knowledge from massive sequences such as document collections, matrix and tensor sequences, or graphs. Among DL models, tensor stack networks (TSNs), implemented in software or in

✉ Georgios Drakopoulos
c16drak@ionio.gr

Eleanna Kafeza
eleana.kafeza@zu.ac.ae

Phivos Mylonas
fmylonas@ionio.gr

Lazaros Iliadis
liliadis@civil.duth.gr

[1] Department of Informatics, Ionian University, Corfu, Greece

[2] College of Technology and Innovation, Zayed University, Dubai, UAE

[3] Laboratory of Mathematics and Informatics, Democritus University of Thrace, Xanthi, Greece

Published online: 09 July 2021

dedicated tensor processing units (TPUs), are becoming popular as they generalize neural networks to inherently handle two-dimensional input [28]. Preprocessing, outlier discovery, dimensionality reduction, or attribute engineering may supplement the main DL methodology to reduce its computational complexity or increase its accuracy [49].

GSP techniques that rely heavily on graphs bring two-dimensional signals over an irregular domain [47]. Once elementary GSP operations such as shifting and sampling are defined, composite ones such as the graph Laplace transform can be constructed. In conjunction with existing efficient computational kernels like the fast Fourier transform (FFT), GSP techniques such as domain transform adaptive filtering algorithms, maximum likelihood estimators (MLE), spectrum estimators, and the discrete cosine transform (DCT) are gradually becoming established analysis tools or preprocessing modules for more sophisticated or data intensive DL schemes.

The primary research objective of this article is a GSP technique for the estimation with a TSN of the topological correlation coefficient between two unweighted graphs compressed with the two-dimensional DCT (DCT2). Thus, the TSN is trained to indirectly invert DCT2 and simultaneously to interpolate knowledge obtained from its training. Said technique has been inspired by domain transform adaptive DSP ones. The proposed scheme differentiates from previous approaches in the way the key elements DCT2 and TSN are combined, as well as in the application to a graph processing setting. The entire scheme has been implemented in Julia.

The remainder of this work is structured as follows. Section 2 briefly reviews the scientific literature regarding neural networks and GSP. The proposed methodology is presented in sections 3 and 4, whereas the results from the benchmark graph sequences are presented in section 5. Future research directions are given in 6. Technical acronyms are explained the first time they are encountered in the text. Tensors and matrices are represented by capital boldface, vectors by small boldface, and scalars by small letters. To keep consistent with GSP notation, vector and matrix indexing will be C style. Finally, Table 1 summarizes the notation of this work.

2 Previous work

GSP is an emerging field where graphs are treated as two-dimensional signals in the time domain defined over an irregular domain [8, 33]. Once elementary operations such as shifting and sampling are defined [46], then transforms to other domains can be built [18, 40]. The graph Fourier transform, based on the eigenexpansion of the respective graph Laplacian matrix, is introduced in [14] and the Laplace transform in [37]. Fundamental concepts for graph processing with neural networks are proposed in [31], whereas a convolutional neural network (CNN) architecture for processing signals expressed with graph connectivity patterns is described in [16]. In generative graph models, graph regularization acts similarly to Tikhonov regularization preventing erroneously graphs from being formed [6, 20]. Graph neural networks (GNNs) constitute a class of iterative methods which directly process topological graph information introduced in [17]. Recent extensive surveys about the field GNNs are [47] and [50].

Tensors represent the next evolutionary step in linear algebra since they allow simultaneous linear couplings between different vector spaces [44]. They have a wide range of applications including adaptive nonlinear system identification [11], face recognition [43], robust multilinear principal component analysis [29], distances for self-organizing maps (SOMs) [12], multiple input multiple output (MIMO) radars [38], and genetic algorithm fitness functions [13]. TSNs have been originally proposed for large vocabulary recognition [48]. Moreover, applications have been extended to sound classification and reconstruction from corrupted files [21] and checking the resiliency of large graphs based on structural patterns such as cycles and triangles [10]. A TSN review is [28].

DCT has a broad spectrum of discrete signal processing (DSP) applications, predominantly in signal compression as shown in [1]. The energy concentration, as well as the properties of the transform domain, is described in [39], while fundamental symmetries allowing fast versions compared to the standard transform definition are explored in [15]. Image watermarking with DCT has been proposed in [7]. The DCT has been applied in fast image fusion hardware architectures as for instance in [42] and the in fusion of visible and infrared images [35]. Recently, a quantum version of DCT for strong medical image encryption and privacy preserving has been developed [34].

Julia is a high-level programming language based on the low-level virtual machine (LLVM) [24, 25] with a syntax influenced from Java [4]. Extensive benchmarkings have shown that Julia has excellent performance despite being an interpreted language [5]. Additionally, it natively supports parallelism and the use of graphics processing units (GPUs) [3]. Specialized packages for Julia cover fields such as operations research [30], mathematical optimization [32], and medical image reconstruction [23].

Table 1 Notation of this article

Symbol	Meaning	First in
$\triangleq$	Equality by definition	Eq. (16)
$\{s_1, \ldots, s_n\}$	Set with $s_1, \ldots, s_n$	Tb. 3
$(t_1, \ldots, t_n)$	Tuple with $t_1, \ldots, t_n$	Eq. (24)
$(t_1, \ldots, t_n)_k$	Tuple with $t_1, \ldots, t_n$ without t_k	Eq. (2)
$\langle s_k \rangle$	Sequence with elements s_k	Sub. 5.1
$\overline{\oplus}$	XNOR Boolean function	Eq. (17)
$\text{prob}\{\Omega\}$	Probability of event Ω	Eq. (23)
$\deg(v)$	Degree of vertex v	Eq. (23)
$\times_k$	Tensor multiplication in the k-th dimension	Eq. (3)
$\|\cdot\|$	Tensor or matrix norm (inferred from context)	Eq. (30)
$\mathbf{I}$	Identity matrix of suitable dimensions	Eq. (14)

3 Tensor stack networks

3.1 Structure

Intuitively speaking TSNs consist of serially connected layers with information moving successively from the input to the output layer. The main TSN structural characteristic is that each layer consists of stacked feedforward neural networks (FFNNs) trained in parallel. The number of layers and FFNNs depends on the specific task and training set size. The TSN architecture has the following properties:

- The synaptic weights of an FFNN are updated with a linear combination of the updates of the others in the cluster. Therefore, each FFNN learns from its own errors and from those made collectively in its layer. This weight correlation leads to smoother differences across a cluster and a more efficient generalization.
- Two-dimensional objects such as matrices, images, or graphs can be directly driven to a TSN without preprocessing entailing vectorization or projection to one dimension, preserving two-dimensional patterns. This additional information can be exploited to approach smoothly two-dimensional objects.
- Two-dimensional items may well represent one-dimensional ones. For instance, matrices can represent clusters with multiple alternative centroids, the vectors closest to the centroid, or the centroids resulting from various averaging functionals. Such representations offer additional flexibility. Moreover, a TSN can be trained to perform multiple simultaneous operations on them.
- There is no specific criterion for selecting the number of levels, as well as their respective compositions. Factors influencing this decision are the number of available training samples, the minimization of model overfitting, the complexity of the training procedure, and the resulting weight sparsity of each level.

Fig. 1 shows a high-level overview of the general architecture of a TSN. Notice that, each layer receives as input not only the predictions of the preceding layer but also the original data. In contrast to regular FFNNs, there is the potential for driving the original network input to each hidden layer with feedforward loops.

For simplicity, the indices of the particular layer have been dropped. Moreover, the description for hidden layers also holds for the output layer with the understanding that matrix $\mathbf{Y}$ contains the final TSN output. The latter is used to compute local discrete error gradients. Each layer comprises of the following elements:

- Matrices $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$ and $\mathbf{S} \in \mathbb{R}^{J_1 \times J_2}$ denote the original and layer input, respectively. Their dimensions need not coincide for any layer after the first.
- The third-order tensors $\mathbf{H} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and $\mathbf{G} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$ contain, respectively, the adjustable weights for $\mathbf{X}$ and $\mathbf{S}$, performing multilinear attribute transformations.
- Matrix $\mathbf{Y} \in \mathbb{R}^{K_1 \times K_2}$ contains the predictions of the layer, namely the output resulting from the activation (or not) of the FFNNs packed in the particular layer.
- The third-order tensor $\mathbf{N} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ contains the clustered FFNNs. Interactions across them are modeled through $\mathbf{Z}_1 \in \mathbb{R}^{N_1 \times N_1 \times N_3}$ and $\mathbf{Z}_2 \in \mathbb{R}^{N_1 \times N_1 \times N_3}$.
- The third-order tensor $\mathbf{V} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ has the values for the individual neurons of every network in the cluster and has the same structure with $\mathbf{N}$.

The orders and the dimensions of these elements are not fixed, excluding the dependencies necessary for operand compatibility. Each layer also may have different orders and dimensions. In the general case, the transformed inputs

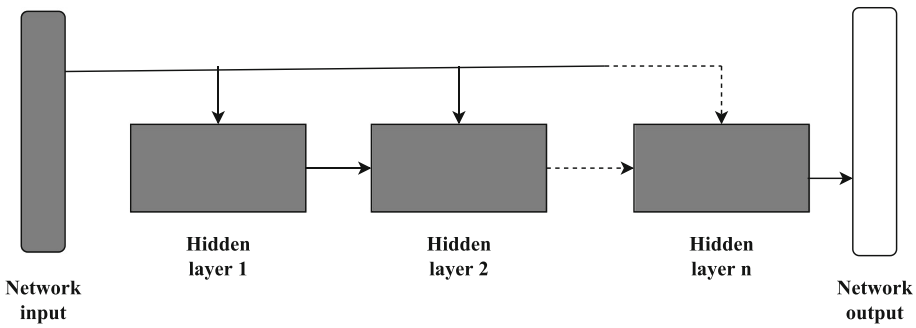

Fig. 1 Architecture of a TSN (Source: Authors)

are driven to a multidimensional neuron grid. Each neuron has a nonlinear activation function $\varphi(\cdot)$ triggered independently. The neuron outputs constitute the layer predictions (Fig. 2).

The above functionality, which captures only the forward pass, takes place as follows. First, the two multilinear transforms shown in Eq. (1) take place:

$$\mathbf{U}_I \triangleq \mathbf{H} \times_1 \mathbf{X} \qquad \text{and} \qquad \mathbf{U}_L \triangleq \mathbf{G} \times_1 \mathbf{S} \tag{1}$$

Matrices $\mathbf{U}_I$ and $\mathbf{U}_L$ are the common but split input layer of the FFNNs packed in $\mathbf{N}$.

The tensor multiplication along the k-th dimension $\times_k$ between the pair of real valued tensors $\mathbf{T}_a \in \mathbb{R}^{I_1 \times \ldots \times I_k \times \ldots \times I_{n_a}}$ of order n_a and $\mathbf{T}_b \in \mathbb{R}^{I'_1 \times \ldots \times I_k \times \ldots \times I'_{n_b}}$ of order n_b is defined elementwise as shown in Eq. (2):

$$\begin{aligned}(\mathbf{T}_a \times_k \mathbf{T}_b)\left[(i_1, \ldots, i_{n_a})_k \left(i'_1, \ldots, i'_{n_b}\right)_k\right] \\ \triangleq \sum_{i_k=1}^{I_k} \mathbf{T}_a[i_1, \ldots, i_{n_a}]\mathbf{T}_b\left[i'_1, \ldots, i'_{n_b}\right]\end{aligned} \tag{2}$$

Note that, $\mathbf{T}_a$ and $\mathbf{T}_b$ must have the same length along the k-th dimension. Moreover, in the left hand side of (2), the two index tuples are concatenated to a single one.

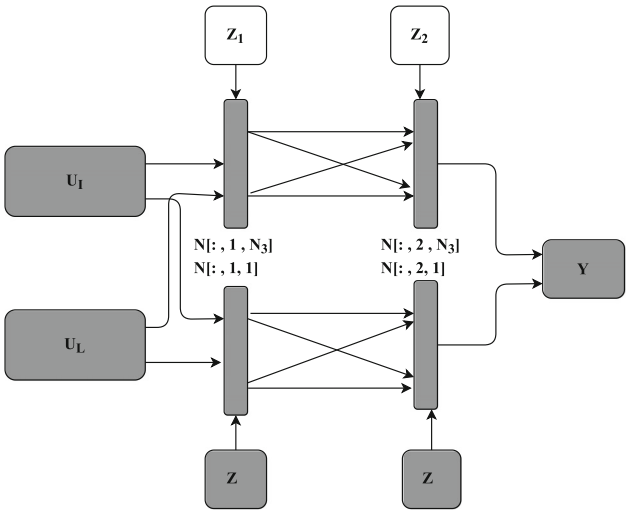

Fig. 2 Hidden layer structure (Source: Authors)

The multiplication between the matrix $\mathbf{X}$, which is a second-order tensor, and the third-order weight tensor $\mathbf{H}$ yields the intermediate input matrix $\mathbf{U}_I$ of Eq. (3):

$$\mathbf{U}_I[i_2, i_3] = (\mathbf{H} \times_1 \mathbf{X})[i_2, i_3] \triangleq \sum_{i_1=1}^{I_1} \mathbf{H}[i_1, i_2, i_3]\mathbf{X}[i_1, i_2] \tag{3}$$

A similar result holds for the intermediate matrix $\mathbf{U}_L$ using the input matrix $\mathbf{S}$ and weight tensor $\mathbf{G}$. This shows the symmetric role of $\mathbf{X}$ and $\mathbf{S}$. Once matrices $\mathbf{U}_I$ and $\mathbf{U}_L$ are computed, they are driven to $\mathbf{N}$, a tensor containing an FFNN cluster and hence the layer computational core. The meaning of each dimension of $\mathbf{N} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ is:

- The first dimension indicates neuron location. Thus, $\mathbf{N}[:, n_2, n_3]$ refers to the weights of layer n_2 of network n_3, whereas $\mathbf{N}[:, :, n_3]$ gives all weights of n_3.
- The second dimension is the layer location of the cluster. Notice that, each layer in every network contains N_1 neurons. In the general case, this need not be true.
- The third dimension identifies an individual network in the cluster. By fixing this index, $\mathbf{N}[:, :, n_3]$ is the synaptic networks internal to the n_3-th FFNN.

The intra-network synaptic weight connections are given from $\mathbf{N}$. Each of the N_3 networks has N_2 layers with N_1 neurons at each layer for a total of N_1N_2 neurons, which is common parameter across all the N_3 networks. The N_1N_3 across all N_3 neurons of the same layer N_2 fire and have their synaptic weights updated together.

The interaction between peer layers, namely layers having the same numbering, is codified in $\mathbf{Z}_1 \in \mathbb{R}^{N_1 \times N_1 \times N_3}$ and $\mathbf{Z}_2 \in \mathbb{R}^{N_1 \times N_1 \times N_3}$. By construction, only peer layers of neighboring FFNNs interact. This has been inspired by the spatial properties of SOMs. Geometrically, each of the N_3 FFNNs of a cluster is placed in the area defined by the first two dimensions of $\mathbf{N}$. Peer interactions between the k-th layers are modeled by $\mathbf{Z}_k$. In the TSN configurations presented here for simplicity and efficiency both in design and in training in each cluster, there are two FFNNs with two layers each. Given $\mathbf{U}_L$ and $\mathbf{U}_I$, the input to the first layer is defined elementwise as in (4):

$$
\begin{aligned}
u \triangleq & \mathbf{N}[n_1, 1, n_3] \prod_{n_2'=1}^{N_2} \mathbf{U}_L[n_1, n_2'] \\
& + (1 - \mathbf{N}[n_1, 1, n_3]) \prod_{n_2'=1}^{N_2} \mathbf{U}_I[n_1, n_2'] \\
& + \sum_{n_1'} \sum_{n_3' \neq n_3} \mathbf{Z}_1[n_1', 1, n_3'] \mathbf{V}[n_1', 1, n_3']
\end{aligned} \tag{4}
$$

This is the input driven to the n_1-th neuron of the first layer of the n_3-th network. Observe it is linear in its arguments and that it consists of the following components:

- A weighted version of $\mathbf{U}_L$ takes into consideration the specific input corresponding to the neuron location in the layer n_1. This creates a shared cluster output.
- A weighted version of $\mathbf{U}_I$. Because of its form, this term competes with the preceding one. Effectively, this forces stacked FFNNs to specialize their outputs.
- The third term is a weighted linear combination of the output of the peer layers, namely the layers with the same numbering, of the neighboring networks.

Once the input u of Eq. (4) is computed, it is driven to the appropriate neuron. Because of its form u depends heavily on the order the values $\mathbf{V}$ are computed inside a given layer. In this article, these values are computed serially based on the location of the FFNN in the cluster and the new values are used wherever possible. For the second layer, the input u' to neuron n_1 is similar to u as shown in Eq. (5):

$$
\begin{aligned}
u' \triangleq & \mathbf{N}[n_1, 2, n_3] \prod_{n_2'=1}^{N_2} \mathbf{V}[n_1, 1, n_2'] \\
& + (1 - \mathbf{N}[n_1, 2, n_3]) \prod_{n_2'=1}^{N_2} \mathbf{U}_I[n_1, n_2'] \\
& + \sum_{n_1'} \sum_{n_3' \neq n_3} \mathbf{Z}_2[n_1', 2, n_3'] \mathbf{V}[n_1', 2, n_3']
\end{aligned} \tag{5}
$$

There are many options for the activation function. The smooth rectified linear unit (sReLU) is a common option shown in (6). As its name suggests, sReLU is a smoother alternative to the original ReLU with sReLU retaining most properties of the latter.

$$
\varphi(x; \sigma_0) \triangleq \ln(e^{\sigma_0 x} + 1), \quad \sigma_0 > 0 \tag{6}
$$

An alternative option is the logistic activation function (lgi) of Eq. (7), which has a natural interpretation in the context of the class of Verhulst population models. In both activation functions of Eqs. (6) and (7), σ_0 is a scaling constant.

$$
\varphi(x; \sigma_0) \triangleq \frac{1}{1 + e^{-\sigma_0 x}} = 1 - \frac{1}{1 + e^{\sigma_0 x}}, \quad \sigma_0 > 0 \tag{7}
$$

The scaled polynomial constant unit (SPOCU) is a third option for the activation function [22]. It relies on the scaled difference between the evaluations in two closely spaced points of a polynomial kernel as shown in (8).

$$
\begin{aligned}
& \varphi(x; \alpha_0, \beta_0.\gamma_0) \triangleq \alpha_0 \left(h\left(\frac{x}{\gamma_0} + \beta_0 \right) - h(\beta_0) \right), \\
& \alpha_0, \gamma_0 > 0, \beta_0 \in (0, 1)
\end{aligned} \tag{8}
$$

The eighth degree polynomial kernel $h(\cdot)$ of Eq. (8) is defined as in (9). The constant c imposes a hard upper limit on the neuron activation function value.

$$
h(x; c) \triangleq \begin{cases} c^3\left(c^5 - 2x^4 + 2\right), & x > c \\ x^3\left(x^5 - 2x^5 + 2\right), & 0 \leq x \leq c, \\ 0, & x < 0 \end{cases} \qquad c \geq 1 \tag{9}
$$

For the proper and efficient training in terms of convergence speed of the TSN, the activation function $\varphi(\cdot)$ should satisfy at least the following fundamental conditions:

- There should be a thresholding behavior of $\varphi(\cdot)$ or a smooth approximation thereof. This is critical for the characteristic nonlinear behavior of the FFNNs inherited by the TSNs. Firing thresholding is also a basic trait of human neurons.
- The local derivative of $\varphi(\cdot)$ for both input matrices should be bounded in order to ensure both smooth weight corrections, as well as a total smooth convergence.

Note that, the logistic function of Eq. (7) satisfies both criteria. On the other hand, the sReLU of Eq. (6) is unbounded. However, because of its linear scaling, which is sufficiently low, it can yield acceptable values for a wide range of σ_0 which do not lead to numerical instability for bounded input. SPOCU is also by construction bounded. All activation functions are evaluated for each neuron individually.

In this particular case, the TSN augments its input with knowledge stored in it. Therefore, it is essential that the number of hidden layers and adjustable parameters be kept to a minimum so that the input information is fully exploited without model overfit. To this end, the number of hidden layers of TSN will be kept low, namely one and two, in the fashion of autoencoders and extreme learning machines (ELMs).

3.2 Training

As a general remark, training a TSN is considerably more complex than an ordinary FFNN. This is attributed

primarily to the estimation at each layer of the interconnected local error gradients of the neuron grid $\mathbf{N}$. The latter must be done correctly both algorithmically and numerically to avoid error magnification through the TSN.

First, however, the error metric L quantifying the distance between the final TSN output and the corresponding desired response should be defined. Since the desired output is the topology correlation coefficient r_d of Eq. (16), it follows immediately that the last layer of the TSN should be also yield a scalar y. In this case, assuming that C_0 results $\{r_{d,k}\}$ are available, then L is defined as in Eq. (10):

$$L \triangleq \frac{1}{C_0}\sum_{k=1}^{C_0}\left(r_{d,k}-y_k\right)^2 \tag{10}$$

This is the deterministic mean square error taken over all the available C_0 pairs of the actual TSN output y_k and the corresponding desired output $r_{d,k}$. This particular is a reasonable choice for both scalar network outputs and the TSN extrapolation task.

The initial synaptic weights of $\mathbf{H}$, $\mathbf{G}$, $\mathbf{Z}_1$, $\mathbf{Z}_2$, and $\mathbf{G}$ at each layer and output values of $\mathbf{V}$ at each level are drawn from the uniform distribution $[-1,+1]$. At each layer, the available synaptic weights, namely the elements of weight tensors $\mathbf{H}$ and $\mathbf{G}$ and of the neuron grid $\mathbf{N}$, are updated in a single pass in the $(k+1)$-th epoch according to the delta rule of Eq. (11) as appropriate.

$$\begin{aligned}\mathbf{H}^{[k+1]} &= \mathbf{H}^{[k]} + \mu_0^{[k+1]}\mathbf{X}^{[k+1]}\left(\nabla_{\mathbf{X}}\mathbf{N}^{[k+1]}\right)^T\nabla_{\mathbf{X}}\mathbf{N}^{[k+1]}\\ \mathbf{G}^{[k+1]} &= \mathbf{G}^{[k]} + \mu_0^{[k+1]}\mathbf{S}^{[k+1]}\left(\nabla_{\mathbf{S}}\mathbf{N}^{[k+1]}\right)^T\nabla_{\mathbf{S}}\mathbf{N}^{[k+1]}\end{aligned} \tag{11}$$

In equation (11), $\mu_0^{[k+1]}$, the learning rate, has been selected as the cosine decay rate defined as in (12), where M_0 is an estimation of the number of training iterations.

$$\mu_0^{[k]} \triangleq \begin{cases}\cos\left(\dfrac{2\pi k}{M_0+1}\right), & 0 \le k \le M_0/4\\ \cos\left(\dfrac{\pi M_0}{2(M_0+1)}\right), & k > M_0/4\end{cases} \tag{12}$$

The delta rule for the synaptic weight update of the $\mathbf{N}$ is given in Eq. (13):

$$\mathbf{N}^{[k+1]} \triangleq \mathbf{N}^{[k]} + \mu_0^{[k+1]}\left(\nabla_{\mathbf{X}}\mathbf{N} + \nabla_{\mathbf{S}}\mathbf{N}\right) \times_1 \mathbf{E}^{[k+1]} \tag{13}$$

The error matrix $\mathbf{E}^{[k+1]}$ of Eq. (13) is given by Eq. (14):

$$\mathbf{E}^{[k]} \triangleq \begin{cases}\mathbf{I} - 2\mathbf{Y}^{[k]T}\varphi\left(\mathbf{Y}^{[k]}\right), & \text{hidden layer}\\ \dfrac{2}{C_0}\left(y_k - r_{d,k}\right), & \text{output layer}\end{cases} \tag{14}$$

In (14), the notation $\varphi(\mathbf{M})$ is the elementwise application of $\varphi(\cdot)$ to matrix $\mathbf{M}$.

The local error gradient with respect to $\mathbf{X}$ of $\mathbf{N}$ and $\mathbf{S}$ for a layer different than the output one is the respective difference of the current weight tensor value from the past one. The weight updates for the tensors $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are given in (15):

$$\begin{aligned}\mathbf{Z}_1^{[k+1]}[n_1,n_2,n_3] &\triangleq \mathbf{Z}_1^{[k]}[n_1,n_2,n_3] + \mu_0^{[k+1]}u\mathbf{E}[n_1,n_2]\\ \mathbf{Z}_2^{[k+1]}[n_1,n_2,n_3] &\triangleq \mathbf{Z}_2^{[k]}[n_1,n_2,n_3] + \mu_0^{[k+1]}u'\mathbf{E}[n_1,n_2]\end{aligned} \tag{15}$$

4 Proposed methodology

4.1 Graph topology correlation

Having described in detail the general case of the functionality of a TSN, the proposed DL scheme and its purpose can now be explained. Comparing the topology of two large-scale unweighted graphs is by no means trivial because of scalability issues. For two graphs without loops with n vertices, each let the correlation coefficient r_d be defined as in Eq. (16) [9]:

$$r_d \triangleq \frac{1}{n^2-n}\sum_{i_1=0}^{n-1}\sum_{i_2=0}^{n-1}\mathbf{M}_1[i_1,i_2]\mathbf{M}_2[i_1,i_2] \tag{16}$$

Equation (16) assumes that a bijection between the two vertex sets has been already established. Although ruled out in many cases, loops may have a valid interpretation in certain domains such as state modeling of transmission attempts over shared media in communication networks [20, 26]. In these cases, only the leading normalization coefficient needs to be changed.

Observe also that r_d counts only common edge occurrences. In certain domains, it may be desirable that common edge absences are also taken into consideration, for instance in link prediction problems. Such problems frequently arise among others in recommendation engines in the form of item suggestion [27] and in social network analysis in the form of friend discovery [41]. In these cases, the correlation coefficient r_a of (17) can be used:

$$r_a \triangleq \frac{1}{n^2-n}\sum_{i_1=0}^{n-1}\sum_{i_2=0}^{n-1}\mathbf{M}_1[i_1,i_2]\bar{\oplus}\mathbf{M}_2[i_1,i_2] \tag{17}$$

In Eq. (17), the operation $\bar{\oplus}$ between the adjacency matrix entries is the complementary XOR Boolean function which takes a value of one when both entries have the same value. In any case, the proposed methodology can be applied to this correlation coefficient variant as well without any loss of generality.

The difference between the two correlation coefficients of Eqs. (16) and (17) can be illustrated with the following

example. In Fig. 3, two input graphs are shown. The corresponding adjacency matrices are given in Eq. (18).

$$\mathbf{M}_1 \triangleq \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{M}_2 \triangleq \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{18}$$

For the specific example, the values are $r_a = 1/3$ and $r_d = 2/3$. The fact that the latter is twice the former is directly attributed to the fact that common edge absences are factored in besides common edge occurrences.

4.2 Graph compression

The first step of the proposed methodology is to compute the DCT2 of the respective adjacency matrix. Given the energy compression property of this transform, keeping a fraction of the largest DCT2 coefficients is tantamount to keeping only the most important higher-order structural graph properties. In turn, this is equivalent to compressing the input graph. For a signal $x \in \mathbb{R}^{I_1 \times I_2}$, the DCT2 is a linear transform to the two-dimensional spatial frequency domain and as stated earlier, it is most commonly associated with numerous DSP applications. The most common DCT2 variant is the so-called type II defined as shown in Eq. (19):

$$X[u_1, u_2] \triangleq \beta_1[u_1]\beta_2[u_2] \sum_{i_1=0}^{I_1-1} \sum_{i_2=0}^{I_2-1} x[i_1, i_2] \cos \omega_1 \cos \omega_2 \tag{19}$$

The spatial frequency variables ω_1 and ω_2 are defined as in Eq. (20):

$$\omega_1 \triangleq \frac{\pi u_1 (2i_1 + 1)}{2I_1} \quad \text{and} \quad \omega_2 \triangleq \frac{\pi u_2 (2i_2 + 1)}{2I_2} \tag{20}$$

The normalization coefficients $\beta_1[u_1]$ and $\beta_2[u_2]$ of Eq. (19), which ensure DCT2 is an isometric transform, take the values shown in Eq. (21):

$$\beta_1[u_1] \triangleq \begin{cases} \sqrt{\frac{2}{I_1}}, & u_1 = 0 \\ \sqrt{\frac{1}{I_1}}, & u_1 \neq 0 \end{cases} \quad \beta_2[u_2] \triangleq \begin{cases} \sqrt{\frac{2}{I_2}}, & u_2 = 0 \\ \sqrt{\frac{1}{I_2}}, & u_2 \neq 0 \end{cases} \tag{21}$$

In a GSP setting, it makes perfect sense to ask what is the meaning of spatial frequency. The latter is inherently tied to graph sampling. A high spatial frequency denotes denser sampling in the sense that fewer intermediate vertices, as determined by the adjacency matrix, are skipped [37]. Based on this interpretation, DCT2 coefficients, especially those in low spatial frequencies, contain a part of the patterns of local paths. Thus, higher-order graph structure is reflected in the DCT2 coefficients. A correlation preserving relationship similar to Parseval's equality ensures that topological comparisons in the original and in the DCT2 domain are equivalent [39]. Furthermore, because of the strong energy concentration, only few of the DCT2 coefficients can capture most of the topological properties of the original graph [15]. This is the reason the DCT and DCT2 are heavily used in popular audio and image compression standards such as JPEG and MP3. Still, in practice, DCT2 has a compression limit beyond which artifacts appear [7]. Moreover, since in this case, the DCT2 coefficients are real, only their magnitude needs to be examined.

The computational complexity of DCT2 is in practice much lower compared to the quadratic number of operations dictated by equation (19). This is explained as DCT2 is the real part of the two-dimensional Fourier transform. Thus, DCT2 can be derived through efficient implementations exploiting trigonometric symmetries, as well as its recursive nature [15]. Concerning memory requirements, DCT2 is *in-place* and so the additional overhead is at most of the order of magnitude of the input data [1].

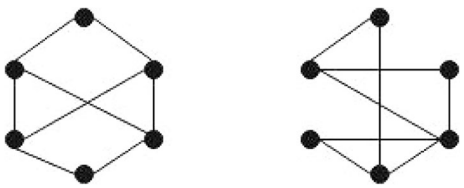

Fig. 3 Two example input graphs (Source: Authors)

4.3 Graph decompression and structural augmentation

Once the DCT2 coefficients have been computed and selected, they are driven to an appropriately configured TSN which acts as an extrapolator or predictor followed by an inverse transformer. The cost of replacing the IDCT2

with the more expensive TSN is justified by the improved prediction of the correlation coefficient (16).

Let n be the number of vertices for the two graphs for which the correlation coefficient is sought. Then, the input to the TSN is a pair of $n \times n$ matrices containing only the c_0 highest DCT2 coefficients in their respective locations and zero at the remaining locations. The number of non-zero coefficients c_0 is $\lceil \gamma_0 n \rceil$.

In the proposed scheme, the packed FFNNs in each layer contain c_0 processing neurons. Thus, each FFNN in the first layer of the TSN has $2N_3c_0$ neurons, and each FFNN at the second layer of the TSN $N_3(c_0 + 1)$. Given that each TSN layer contains N_3 packed networks, the total number of neurons N_p in the TSN is given by (22):

$$N_p = N_3(3c_0 + 1) \tag{22}$$

This is pseudolinear in the number of the DCT2 coefficients, meaning that if the number of packed FFNNs in each TSN layer N_3 is $o(c_0)$, then the N_0 is linear in c_0. Therefore, in this case, the space complexity of the TSN is comparable to that of the DCT2. Moreover, regarding the total number of operations required by the TSN, the following can be said about the interaction of a single neuron at each iteration:

- Each interacts with at most c_0 others at the next layer of the same FFNN.
- Also, it interacts with at most $2c_0$ others at peer layers of neighboring FFNNs.

Given the above, the total number of interactions per iteration N_i is $3c_0^2$. When the number of training iterations is sublinear in c_0, then the overall complexity is comparable to that of the standard DCT2 but higher than that of the fast DCT2. Nevertheless, for a sequence of graphs or for large and sparse graphs, the proposed scheme may be a viable alternative to the IDCT2 from a complexity perspective.

Recall that an epoch is defined as the number of iterations required to feed the TSN with every training sample once. Thus, the number of iterations equals by definition the number of the samples. Assuming a total of g_0 available samples, let the $\lceil t_0 g_0 \rceil$ be reserved for training purposes, $\lceil t_1 g_0 \rceil$ for testing purposes, and the remaining for validation once the TSN configurations are tested.

DCT2 has been consistently reported as achieving better compression ratios under a relatively broad spectrum of conditions. This implies that the total number of levels and their synaptic weights can be kept low. Moreover, TSNs can offer additional generalization power in reconstructing two-dimensional objects compared to the DCT2 compression alone since the synaptic weights, if properly trained, can learn to augment the input data with semantic information in order to compute the correlation coefficient with a much higher accuracy. In graphs and images, information is typically stored in a distributed manner, resulting in a high semantic similarity between most neighboring vertices and pixels, respectively. However, the former represent an irregular domain in the sense that not only the number of connections may well vary across vertices, but also local connectivity patterns. Therefore, graphs have richer structural patterns compared to an image.

5 Results

5.1 Experimental setup

In order to experimentally evaluate the proposed methodology in terms of accuracy and complexity, three undirected graphs were downloaded from the *network repository* [36]. Specifically, the *inf-power* [45], the *yeast-protein-inter* [19], and the *ca-Erdos992* [2] graphs were selected, forming, respectively, the graphs P_0, Y_0, and E_0. For each of these, the corresponding graph sequences $\langle P_k \rangle$, $\langle Y_k \rangle$, and $\langle E_k \rangle$ were generated, each containing g_0 graphs, using the following probabilistic procedure:

- With probability p_0, select uniformly one vertex from the $\lceil v_0 n \rceil$ ones having the lowest degrees, where n is the total number of vertices. Also, select uniformly one vertex from the $\lceil v_0 n \rceil$ ones having the highest degrees and connect them.
- Otherwise, with probability $1 - p_0$, select two vertices with probability proportional to their locally weighted degrees as shown in (23) and connect them.

$$\text{prob}\{v\} \propto \frac{\deg(v)}{\sum_u \deg(u)} \tag{23}$$

In the above equation, the denominator ranges over all neighbors of v. This mechanism prevents all high degree vertices of obtaining immediately more neighbors, essentially dominating the graph. This exponential degree growth is not observed in large real-world graphs. On the contrary, the latter have rather sizeable groups of vertices with moderate degrees acting as local hubs. Equation (23) achieves that approximately. Notice that, all graphs in the sequence have the same number of vertices.

The configuration of each TSN in the experiments is represented by a tuple (24):

$$(\text{input\_size}, \text{activation\_function}) \tag{24}$$

Table 2 contains the TSN configurations used. The left column contains configurations with the sReLU activation function, whereas the right one has configurations with the lgi. Configurations in the same row differ only in the activation function.

Table 2 essentially represents a non-exhaustive search of the parameter space. The points by definition represent a valid individual TSN architecture. Comparing them sheds light into the relative strengths and weaknesses of each.

Table 3 contains all the parameters which have been mentioned in the preceding analysis along with their respective values for convenience.

5.2 TSN configuration evaluation

Reconstructing a graph is similar to image or music recovery. In the graph topology context, the DCT coefficients can be interpreted as views of the graph of variable resolution. To see why, consider first the DCT coefficients of lower spatial frequencies which represent in a more compact form the higher-order global structural patterns of the topology. Adding more coefficients with progressively higher spatial frequencies leads to gradually more refined views of the graph, until the last coefficients which represent local patterns add the finishing touch in graph reconstruction. So, keeping only a fraction of the coefficients is tantamount to lossy topology compression.

The preceding qualitative analysis raises the two fundamental questions of how many coefficients are sufficient for a given level of information loss and how the latter is defined. In several cases [39], the ranked DCT coefficients in terms of amplitude X_k have been reported to have a power law decay as in Eq. (25):

$$X_k = \alpha_0 (k+1)^{-\eta_0}, \quad \alpha_0, \eta_0 > 0, \quad 0 \le k \le n^2 - 1 \tag{25}$$

Taking the natural logarithm of (25) for each k and stacking the n^2 equations yield the linear system of Eq. (26). Observe that its coefficient matrix is tall.

$$\begin{bmatrix} \ln X_0 \\ \ln X_1 \\ \vdots \\ \ln X_{n^2-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & -\ln 2 \\ \vdots & \vdots \\ 1 & -2\ln n \end{bmatrix} \begin{bmatrix} \ln \alpha_0 \\ \eta_0 \end{bmatrix} \Leftrightarrow \mathbf{x} = \mathbf{A}\mathbf{w} \tag{26}$$

Since the system of equation (26) does not have a unique solution, an approximation thereof can be sought. One such solution is the least squares (LS) of (27):

$$\mathbf{w}_{LS} \triangleq \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{x} \tag{27}$$

This imples that the values predicted by the LS solution are given by (28):

$$\mathbf{x}_{LS} = \mathbf{A}\mathbf{w}_{LS} = \mathbf{A}\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{w} = \mathbf{P}\mathbf{w} \tag{28}$$

Moreover, observe that the matrix $\mathbf{P}$ is a orthogonal projection matrix since:

$$\mathbf{P}^2 = \mathbf{A}\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T \cdot \mathbf{A}\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T = \mathbf{P} \tag{29}$$

The relative residual B is a figure of merit of the LS goodness of fit shown in (30). It measures the ratio of the amount of energy of the original measurements contained in the residual space, which is perpendicular and hence inaccessible to LS solution, to the amount of energy retained by the LS solution. This exploits the fact that the operators $\mathbf{I} - \mathbf{P}$ and $\mathbf{P}$ span two orthogonal spaces the union of which has $\mathbf{x}$.

$$B \triangleq \frac{\left\| \left(\mathbf{I} - \mathbf{A}\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\right)\mathbf{x} \right\|_2}{\left\| \mathbf{A}\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{x} \right\|_2} = \frac{\|(\mathbf{I} - \mathbf{P})\mathbf{x}\|_2}{\|\mathbf{P}\mathbf{x}\|_2} \tag{30}$$

Since one is interested in retaining as much of the original power in the LS solution, it follows that the lower B is, the better the LS solution is—with zero being the obvious lowest possible bound. Table 4 shows B, as well as the minimum, maximum, and average exponent over each member of each sequence. Also, it shows the average fraction $\bar{\gamma}_0$ of the DCT2 coefficients necessary to capture the τ_0 of the total coefficient energy. This has been obtained by averaging the individual fractions for member in the respective sequence. The values of $\bar{\gamma}_0$ justify the values of γ_0 in Table 3. The minimum, maximum, and

Table 2 TSN configurations

#	Configuration	#	Configuration	#	Configuration
1	$(0.1P_0, sReLU)$	6	$(0.1P_0, lgi)$	11	$(0.1P_0, SPOCU)$
2	$(0.15P_0, sReLU)$	7	$(0.15P_0, lgi)$	12	$(0.15P_0, SPOCU)$
3	$(0.2P_0, sReLU)$	8	$(0.2P_0, lgi)$	13	$(0.2P_0, SPOCU)$
4	$(0.25P_0, sReLU)$	9	$(0.25P_0, lgi)$	14	$(0.25P_0, SPOCU)$
5	$(0.3P_0, sReLU)$	10	$(0.3P_0, lgi)$	14	$(0.3P_0, SPOCU)$

Table 3 Experimental setup

Parameter	Value
percentage of coefficients γ_0	$\{0.1, 0.15, 0.2, 0.25, 0.3\}$
graphs in each sequence g_0	10000
direct selection probability p_0	0.05
fraction of low degree vertices v_0	0.15
training set percentage t_0	0.6
testing set percentage t_1	0.3
learning rate μ_0 decay	cosine - eq. (12)
number of training iterations M_0	128
runs for each configuration R_0	1000
DCT2 coefficient power threshold τ_0	0.75

average densities ρ_0 for each sequence are given as well. Recall that the density is the number of edges to the number of vertices.

Since the B is low and the exponent range is narrow, it can be assumed that in all three cases, the DCT2 coefficients decay with a power law. The relationship between the number of coefficients for a given power threshold τ_0 can be approximated as follows. Summing the DCT2 coefficients yields Eq. (31):

$$\sum_{k=1}^{\lceil \gamma_0 n \rceil} \alpha_0 k^{-\eta_0} \approx \alpha_0 \int_1^{c_0} x^{-\eta_0} dx = \alpha_0 \left(1 - \frac{c_0^{1-\eta_0}}{1-\eta_0} \right) \leq \tau_0 \tag{31}$$

Given a power threshold τ_0 for the DCT2 coefficients, c_0 and γ_0 can be computed.

The TSN configurations of the proposed scheme will be evaluated for each benchmark graph sequence. This evaluation is based on the following criteria:

- The normalized MSE of the test set for the respective TSN configuration E_t.
- The normalized number of epochs for each TSN configuration N_e.
- The ratio of the execution time to that of computing r_d of a graph pair T_e.
- The same without the training time for the TSN configurations T_c.

The first three criteria aim at comparing the TSN configurations with each other. The normalized values have been selected as they yield better insight on the configuration performance by providing relative measurements. The next two criteria focus on the comparison between the best TSN configurations and the direct method of compressing two graphs with the DCT2 and computing their correlation coefficient. Thus, they reveal whether it is worth trading time for a better estimation of r_d. Perhaps more interestingly, the last criterion aims at showing whether the proposed scheme is a viable alternative for graph sequences. The results are summarized in Table 5.

Given the entries of Table 5, the following can be said:

- Configurations 8, 9, and 10 systematically yield better performance. All three utilize the lgi activation function, which is highly nonlinear compared to the sReLU, and rely on a large number of DCT2 coefficients.
- The more coefficients are available, the lower the error is. This is expected as more coefficients offer a finer topology view. As a limiting case, taking all DCT2 coefficients is equivalent to computing the correlation coefficient.
- There appears to be a tradeoff between lowering the error and the number of iterations in the above three TSN configurations. This also applies when these configurations are compared to the compressed method.
- Taking fewer DCT2 coefficients also leads in an increased number of iterations. This is attributed to the slower convergence rate, as well as to the higher steady-state error caused by the limited attributes available to the TSN.
- The lgi outperforms sReLU and SPOCU with the latter being the second best. This can be attributed to the strong nonlinearity of lgi and SPOCU, as well as to the monotonicity of lgi which is suitable for the interpolation task of the TSN.
- TSNs can learn easier sparser graphs than denser ones. This can be attributed to the fact that the former have less patterns to learn and regenerate at fewer locations. The DCT2 can capture easier the structure of sparse graphs.
- The proposed scheme is efficient compared to the compressed method in terms of computational time for large graphs, especially for sparse ones, only when it is applied to many graph pairs once the TSN is trained.

DCT2 is a standard DSP algorithm and it can be inverted in many ways. Using a TSN for this particular purpose is a novelty driven by the following reasons:

- TSNs can achieve robust generalization from the training graphs and, provided the latter are structurally strongly related to the input graphs, lower the error.
- In an evolving environment such as social networks or supply chains, TSNs can track its dynamics by synaptic network update through short training graphs.

Although the computational complexity of r_d in Eq. (16) can be clearly described by a closed formula, that of the proposed scheme is not. This can be attributed to the lack of a reliable prediction mechanism for the number of training iterations of a TSN. To this end, as stated earlier, the normalized and averaged over R_0 runs number of

Table 4 LS residual error, exponents, and density

Seq.	B	min η_0	max η_0	avg η_0	$\bar{\gamma}_0$	min ρ_0	max ρ_0	avg ρ_0
$\langle P_k \rangle$	0.0152	−2.5667	−2.2551	−2.3216	0.2885	27.5688	84.2331	55.9833
$\langle Y_k \rangle$	0.0542	−2.8854	−2.5993	−2.6781	0.2566	33.6667	96.4411	61.1120
$\langle E_k \rangle$	0.0433	−2.9432	−2.6883	−2.8563	0.2319	48.4551	112.3334	74.4453

Table 5 Results for each TSN configuration and benchmark sequence

#	$\langle P_k \rangle$				$\langle Y_k \rangle$				$\langle Y_k \rangle$			
	E_t	N_e	T_e	T_c	E_t	N_e	T_e	T_c	E_t	N_e	T_e	T_c
1	2.33	1.92	1.83	2.03	2.13	2.11	1.92	1.86	1.91	1.88	2.15	2.03
2	2.13	1.62	1.77	1.94	2.01	1.95	1.75	1.72	1.80	1.72	2.03	1.84
3	1.88	1.41	1.63	1.78	1.89	1.71	1.73	1.63	1.76	1.74	1.81	1.75
4	1.69	1.23	1.58	1.61	1.78	1.63	1.59	1.43	1.51	1.62	1.67	1.61
5	1.62	1.17	1.61	1.59	1.61	1.67	1.37	1.62	1.43	1.49	1.51	1.43
6	1.13	1.09	1.19	1.13	1.11	1.08	1.09	.141	1.05	1.09	1.12	1.18
7	1.09	1.04	1.07	1.08	1.08	1	1.13	1.14	1.17	1.14	1.09	1.11
8	1	1.02	1	1	1	1.03	1	1.04	1	1.03	1	1
9	1.04	1	1.12	1.14	1.13	1.12	1.17	1	1.14	1.17	1.18	1.21
10	1.07	1.09	1.19	1.22	1.23	1.19	1.24	1.12	1.17	1.21	1.25	1.26
11	1.38	1.36	1.32	1.28	1.39	1.27	1.43	1.29	1.28	1.42	1.45	1.47
12	1.46	1.42	1.38	1.43	1.47	1.34	1.51	1.38	1.39	1.47	1.54	1.56
13	1.35	1.28	1.22	1.25	1.27	1.21	1.29	1.21	1.22	1.24	1.25	1.27
14	1.44	1.42	1.41	1.46	1.44	1.45	1.42	1.46	1.45	1.42	1.45	1.47
15	1.53	1.47	1.54	1.52	1.56	1.53	1.52	1.51	1.54	1.55	1.53	1.56

iterations has been recorded. Since the TSN is computationally intensive and CPU bound, the number of iterations is a good indicator of the true execution time.

As for the TSN training overhead, it occurs only once and then the network can be used as a black box. Moreover, in the scenarios examined here, both the number of layers and the number of weights are kept intentionally small.

5.3 Discussion

The outcome of the experiments agrees with the previously reported results. The power law decay of the DCT coefficient amplitudes is mentioned in [39] and [15]. Moreover, the DCT is a preprocessing or nonlinear attribute engineering step in DL pipelines as explained among others in [49].

The preceding analysis showed the broad generality of the proposed scheme as it did not rely on a particular context. Nevertheless, potential applications include:

- In a computer communication, network topologies are an integral part, especially if multiple protocols such as the user datagram protocol (UDP) or the transmission control protocol (TCP), each with its own network layout, are utilized. The proposed scheme can discover bottlenecks or propose packet routing alternatives.
- In a signal processing context, graph topologies may represent mobile terminal locations. Given past location patterns, the proposed scheme can yield resource allocation profiles. In a MIMO context, the TSN can adaptively equalize effects caused by the physical propagation medium such as multipath and refraction.
- In social networks, the proposed methodology can discover latent patterns or augment account topologies with context information. As a concrete example, in a Twitter subgraph comprising of *follow* relationships, probable influencers may be discovered or its overall sentiment with respect to an event may be estimated.
- In a probabilistic topological machine learning (ML) scenario, the proposed scheme can find similar topologies with better properties such as advanced interpretability or lower computational complexity. When training ML models, different policies may be compared for finding the one allowing maximum parallelism.

As seen before, the advantages of the proposed method are the low mean square error when approximating the correlation coefficient, scalability in terms of the input graph size, the reliance on higher-order graph patterns, and the independence from any underlying domain semantics. On the other hand, this generality comes at the cost of ignoring possible additional functional information not captured in the graph structure. Moreover, in the general case, the large number of synaptic weights may slow down the TSN training process and a suitable pruning method has to be applied.

6 Conclusions and future work

This article presents a novel graph signal processing technique for estimating the topological correlation coefficient of two compressed graphs. In particular, a tensor stack network (TSN) is trained to estimate this coefficient

between the two-dimensional discrete cosine transforms (DCT2) of two graph adjacency matrices. Thus, the TSN indirectly acts simultaneously as decoder and interpolator of already obtained knowledge. Although the latter is a well-known architecture and DCT2 is an established signal processing algorithm, to the best of the knowledge of the authors, they have neither been combined before nor have they been applied to graphs. The major advantages of the proposed methodology are that it can be applied to large and potentially sparse graphs or graph sequences, it attains low mean square error compared to the direct correlation coefficient computation, it is independent of the semantics of the underlying domain, and it focuses on higher-order graph structural patterns. Moreover, for evolving graph sequences, the computational complexity is low. The above were obtained from experiments conducted with three benchmark graph sequences over various configurations of the proposed scheme.

Regarding future research directions, exploring the scalability limits of the proposed scheme warrants further investigation. Additionally, techniques for taking into consideration partial, fuzzy, or incomplete topologies should be explored. Finally, the connection between the density in the input topologies and that of the synaptic weights should be investigated and potentially exploited in pruning methods.

Acknowledgements This article is part of Project 451, a long-term research initiative whose primary objective is the development of novel, scalable, numerically stable, and interpretable tensor analytics.

Declarations

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

References

1. Ahmed N, Natarajan T, Rao KR (1974) Discrete cosine transform. IEEE Trans Comput 100(1):90–93
2. Batagelj V, Mrvar A (2000) Some analyses of Erdos collaboration graph. Soc Netw 22(2):173–186
3. Besard T, Foket C, De Sutter B (2018) Effective extensible programming: unleashing Julia on GPUs. IEEE Trans Parallel Distrib Syst 30(4):827–841
4. Bezanson J, Edelman A, Karpinski S, Shah VB (2017) Julia: a fresh approach to numerical computing. SIAM Rev 59(1):65–98
5. Bezanson J, Chen J, Chung B, Karpinski S, Shah VB, Vitek J, Zoubritzky L (2018) Julia: dynamism and performance reconciled by design. ACM Conf Program Lang 2(OOPSLA):1–23
6. Buccini A, Pasha M, Reichel L (2020) Generalized singular value decomposition with iterated Tikhonov regularization. J Comput Appl Math. https://doi.org/10.1016/j.cam.2019.05.024
7. Byun SW, Son HS, Lee SP (2019) Fast and robust watermarking method based on DCT specific location. IEEE Access 7:100706–100718
8. Chen S, Varma R, Sandryhaila A, Kovačević J (2015) Discrete signal processing on graphs: sampling theory? IEEE Trans Signal Process 63(24):6510–6523
9. Drakopoulos G, Kafeza E (2020) One dimensional cross-correlation methods for deterministic and stochastic graph signals with a Twitter application in Julia. SEEDA-CECNSM. https://doi.org/10.1109/SEEDA-CECNSM49515.2020.9221815
10. Drakopoulos G, Mylonas P (2020) Evaluating graph resilience with tensor stack networks: a keras implementation. NCAA 32(9):4161–4176. https://doi.org/10.1007/s00521-020-04790-1
11. Drakopoulos G, Mylonas P, Sioutas S (2019) A case of adaptive nonlinear system identification with third order tensors in TensorFlow. INISTA. https://doi.org/10.1109/INISTA.2019.8778406
12. Drakopoulos G, Giannoukou I, Mylonas P, Sioutas S (2020) On tensor distances for self organizing maps: clustering cognitive tasks. DEXA. Springer, Cham, pp 195–210
13. Drakopoulos G, Stathopoulou F, Kanavos A, Paraskevas M, Tzimas G, Mylonas P, Iliadis L (2020) A genetic algorithm for spatiosocial tensor clustering. EVOS 11(3):491–501. https://doi.org/10.1007/s12530-019-09274-9
14. Ekambaram VN, Fanti GC, Ayazifar B, Ramchandran K (2013) Circulant structures and graph signal processing. In: ICIP, pp 834–838. IEEE
15. Feig E, Winograd S (1992) Fast algorithms for the discrete cosine transform. IEEE Trans Signal Process 40(9):2174–2193
16. Gama F, Marques AG, Leus G, Ribeiro A (2018) Convolutional neural network architectures for signals supported on graphs. IEEE Trans Signal Process 67(4):1034–1049
17. Gori M, Monfardini G, Scarselli F (2005) A new model for learning in graph domains. IJCNN 2:729–734
18. Huang W, Bolton TA, Medaglia JD, Bassett DS, Ribeiro A, Van De Ville D (2018) A graph signal processing perspective on functional brain imaging. Proceedings of the IEEE 106(5):868–885
19. Jeong H, Mason S, Barabasi A, Oltvai Z (2001) Lethality and centrality in protein networks. arXiv preprint cond-mat/0105306
20. Jia Z, Yang Y (2020) A joint bidiagonalization based iterative algorithm for large scale general-form Tikhonov regularization. Appl Numer Math 157:159–177
21. Khamparia A, Gupta D, Nguyen NG, Khanna A, Pandey B, Tiwari P (2019) Sound classification using convolutional neural network and tensor deep stacking network. IEEE Access 7:7717–7727
22. Kisel'ák J, Lu Y, Švihra J, Szépe P, Stehlik M (2021) SPOCU: scaled polynomial constant unit activation function. NCAA 33(8):3385–3401
23. Knopp T, Szwargulski P, Griese F, Grosser M, Boberg M, Möddel M (2019) MPIReco. jl: Julia package for image reconstruction in MPI. Int J Magn Part Imag. https://doi.org/10.18416/IJMPI.2019.1907001
24. Lattner C (2008) LLVM and Clang: next generation compiler technology. The BSD conference, BSD Foundation 5:1–33
25. Lattner C, Adve V (2004) LLVM: a compilation framework for lifelong program analysis & transformation. In: CGO, pp 75–86. IEEE
26. Li Z, Chen J (2017) Robust consensus of linear feedback protocols over uncertain network graphs. IEEE Trans Autom Control 62(8):4251–4258
27. Lian D, Xie X, Chen E (2019) Discrete matrix factorization and extension for fast item recommendation. IEEE Trans Knowl Data Eng 33:1919–1933
28. Liang L, Xu J, Deng L, Yan M, Hu X, Zhang Z, Li G, Xie Y (2021) Fast search of the optimal contraction sequence in tensor networks. IEEE J Select Topics Signal Process 15:574–586

29. Lu C, Feng J, Chen Y, Liu W, Lin Z, Yan S (2019) Tensor robust principal component analysis with a new tensor nuclear norm. IEEE Trans Pattern Anal Mach Intell 42(4):925–938
30. Lubin M, Dunning I (2015) Computing in operations research using Julia. INFORMS Journal on Computing 27(2):238–248
31. Micheli A (2009) Neural network for graphs: a contextual constructive approach. IEEE Trans Neural Netw 20(3):498–511
32. Mogensen PK, Riseth AN (2018) Optim: a mathematical optimization package for Julia. J Open Source Softw. https://doi.org/10.21105/joss.00615
33. Ortega A, Frossard P, Kovačević J, Moura JM, Vandergheynst P (2018) Graph signal processing: overview, challenges, and applications. Proc IEEE 106(5):808–828
34. Pang CY, Zhou RG, Hu BQ, Hu W, El-Rafei A (2019) Signal and image compression using quantum discrete cosine transform. Inf Sci 473:121–141
35. Paramanandham N, Rajendiran K (2018) Infrared and visible image fusion using discrete cosine transform and swarm intelligence for surveillance applications. Infrared Phys Technol 88:13–22
36. Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. In: AAAI, URL http://networkrepository.com
37. Sandryhaila A, Moura JM (2014) Discrete signal processing on graphs: frequency analysis. IEEE Trans Signal Process 62(12):3042–3054
38. Shi J, Wen F, Liu T (2020) Nested MIMO radar: coarrays, tensor modeling and angle estimation. IEEE Trans Aerosp Electron Syst 57:573–585
39. Strang G (1999) The discrete cosine transform. SIAM Rev 41(1):135–147
40. Tremblay N, Amblard PO, Barthelmé S (2017) Graph sampling with determinantal processes. In: EUSIPCO, pp 1674–1678. IEEE
41. Verma J, Gupta S, Mukherjee D, Chakraborty T (2019) Heterogeneous edge embedding for friend recommendation. ECIR. Springer, Cham, pp 172–179
42. Wang M, Shang X (2020) A fast image fusion with discrete cosine transform. IEEE Signal Process Lett 27:990–994
43. Wang SJ, Yang J, Zhang N, Zhou CG (2011) Tensor discriminant color space for face recognition. IEEE Trans Image Process 20(9):2490–2501
44. Wang X, Che M, Wei Y (2019) Neural networks based approach solving multi-linear systems with M-tensors. Neurocomputing 351:33–42
45. Watts DJ, Strogatz SH (1998) Collective dynamics of small-world networks. Nature 393(6684):440–442
46. Wu Y, Cao N, Archambault D, Shen Q, Qu H, Cui W (2016) Evaluation of graph sampling: a visualization perspective. IEEE Trans Vis Comput Graphics 23(1):401–410
47. Wu Z, Pan S, Chen F, Long G, Zhang C, Philip SY (2020) A comprehensive survey on graph neural networks. IEEE Trans Neural Netw Learn Syst 32:4–24
48. Yu D, Deng L, Seide F (2012) The deep tensor neural network with applications to large vocabulary speech recognition. IEEE Trans Audio, Speech, Lang Process 21(2):388–396
49. Zhang C, Cai M, Zhao X, Wang D (2021) Research on case preprocessing based on deep learning. Concurr Comput Pract Exper. https://doi.org/10.1002/cpe.6214
50. Zhou J, Cui G, Zhang Z, Yang C, Liu Z, Wang L, Li C, Sun M (2018) Graph neural networks: a review of methods and applications. arXiv preprint 81208434