RECURRENT GRAPH TENSOR NETWORKS

Yao Lei Xu, Danilo P. Mandic

Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ, UK
E-mails: \{yao.xu15, d.mandic\}@imperial.ac.uk

ABSTRACT

Recurrent Neural Networks (RNNs) [1] are among the most successful machine learning models for sequence modelling. In this paper, we show that the modelling of hidden states in RNNs can be approximated through a multi-linear graph filter, which describes the directional flow of temporal information. The derived multi-linear graph filter is then generalized in tensor network form to improve its modelling power, resulting in a novel Recurrent Graph Tensor Network (RGTN). To validate the expressive power of the derived network, several variants of RGTN models were proposed and employed to the task of time-series forecasting, demonstrating superior properties in terms of convergence, performance, and complexity. Specifically, by leveraging the multi-modal nature of tensor networks, RGTN models were able to outperform a simple RNN by 45% in terms of mean-squared-error while using up to 90% less parameters. Therefore, by combining the expressive power of tensor networks with a suitable graph filter, we show that the proposed RGTN models can outperform a classical RNN at a drastically lower parameters complexity, especially in the multi-modal setting.

Index Terms— Tensor Networks, Tensor Decomposition, Graph Signal Processing, Recurrent Neural Networks, Graph Neural Networks.

1. INTRODUCTION

Tensors are multi-linear generalization of vectors and matrices to multi-way arrays, which allows for a richer representation of the data that is not limited to the classical “flat-view” matrix [5]. Recent developments in tensors have led to Tensor Decomposition (TD) techniques that can represent high dimensional tensors through a contracting network of smaller core tensors. TD techniques can be used to compress the number of parameters needed to represent high-dimensional data, and have found many applications in deep learning. Notably, it has been shown that TD techniques, such as Tensor-Train Decomposition (TTD) [6], can be used to compress neural networks considerably while maintaining comparable performance [2].

The field of Graph Signal Processing (GSP) generalizes traditional signal processing concepts to irregular graph domains [7]. Developments in GSP has led to series of spatial and spectral based techniques that generalise the notion of frequency and locality to irregular domains, allowing for the processing of signals that takes into account the underlying data domain [8]. Several concepts developed in GSP have found application in neural networks, where graph filters can be implemented across multiple layers to incorporate graph information [4].

However, despite the promising results achieved in both fields, the full potential arising from the combination of tensors and graphs is yet to be explored, especially in the area of sequence modelling. To this end, we set out to investigate RNNs using the theoretical framework underpinning tensor networks and graph signal processing. Specifically, we show that the modelling of RNN hidden states can be approximated through a multi-linear graph filtering operation, which can be used in conjunction with tensor networks to create a novel Recurrent Graph Tensor Network (RGTN). Our experimental results confirm the superiority of the proposed RGTN models, demonstrating desirable properties in terms of convergence, performance, and complexity.

We organize the rest of the paper as follows. Section 2 introduces the necessary theoretical background regarding tensors, graphs, and RNNs. Section 3 derives the proposed RGTN models. Section 4 analyses the experimental results achieved by the proposed models, demonstrating their effectiveness. Finally, section 5 summarises the results achieved in the paper.
2. THEORETICAL BACKGROUND

A short theoretical background necessary for this paper is presented below, treating several topics in tensor networks, graph signal processing, and recurrent neural networks. Please refer to [5], [7], and [1] for an in-depth investigation of the subjects.

2.1. Tensors and Tensor Networks

\[ \mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N} \]

- \( N \)-th order tensor of size \( I_1 \times I_2 \times \cdots \times I_N \)

\[ \mathbf{X} \in \mathbb{R}^{I_1 \times I_2} \]

- Matrix of size \( I_1 \times I_2 \)

\[ x \in \mathbb{R} \]

- Vector of size \( I_1 \)

\[ x_{i_1i_2\ldots i_N} = \mathbf{X}(i_1, i_2, \ldots, i_N) \]

- \((i_1, i_2, \ldots, i_N)\) entry of \( \mathbf{X} \)

| Table 1. Tensor, matrix, vector, and scalar notation. |
|------------------------------------------------------|
| An order \( N \) tensor, \( \mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N} \), is an \( N \)-way array with \( N \) modes, where the \( n \)-th mode is of size \( I_n \), for \( n = 1, 2, \ldots, N \). Scalars, vectors, and matrices are special tensors of order 0, 1, and 2 respectively, as detailed in table [1]. A tensor can be reshaped into a matrix through a process known as matricization, while the reverse process is referred to as tensorization [5]. A tensor can also be reshaped into a vector through the vectorization process, which is denoted with the operator vec(·). The tensor indices in this paper are grouped according to the Little-Endian convention [9]. An \((m, n)\)-contraction denoted by \( \times_m \), between an \( N \)-th order tensor \( \mathbf{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) and an \( M \)-th order tensor \( \mathbf{B} \in \mathbb{R}^{J_1 \times \cdots \times J_M} \) with equal dimensions \( I_n = J_m \), yields a third tensor of order \( (N + M - 2) \)

\[ c_{i_1 \cdots i_{n-1}i_{n+1} \cdots i_Nj_1 \cdots j_{m-1}j_{m+1} \cdots j_M} = \sum_{i_n=1}^{I_n} a_{i_1 \cdots i_{n-1}i_ni_{n+1} \cdots i_N} b_{j_1 \cdots j_{m-1}i_nj_{m+1} \cdots j_M} \]  

(1)

A (left) Kronecker product between two tensors \( \mathbf{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) and \( \mathbf{B} \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) denoted by \( \otimes \), yields a tensor of the same order \( \mathbf{C} \in \mathbb{R}^{I_1J_1 \times \cdots \times I_NJ_N} \), with entries \( c_{i_1j_1, \ldots, i_Nj_N} = a_{i_1, \ldots, i_N}b_{j_1, \ldots, j_N} \), where \( i_nj_n = j_n + (i_n - 1)I_n \) \([5]\). For the special case of matrices \( \mathbf{A} \in \mathbb{R}^{I_1 \times I_2} \) and \( \mathbf{B} \in \mathbb{R}^{I_1 \times I_2} \), the Kronecker product yields a block-matrix as:

\[ \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11} \mathbf{B} & \cdots & a_{12} \mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{21} \mathbf{B} & \cdots & a_{22} \mathbf{B} \end{bmatrix} \]  

(2)

2.2. Graph Signal Processing

A graph \( \mathcal{G} = \{ \mathcal{V}, \mathcal{E} \} \) is defined by a set of \( N \) vertices (or nodes) \( \mathcal{V} = \mathcal{V} \), and a set of edges connecting the \( n \)-th and \( m \)-th vertices \( e_{nm} = (v_n, v_m) \in \mathcal{E} \), for \( n = 1, \ldots, N \) and \( m = 1, \ldots, N \). A signal on a given graph is a defined by a vector \( \mathbf{f} \in \mathbb{R}^N \) such that \( \mathbf{f} : \mathcal{V} \rightarrow \mathbb{R} \), which associates a signal value for every node on the graph [12].
A given graph can be fully described in terms of its weighted adjacency matrix $A \in \mathbb{R}^{N \times N}$ such that $a_{nm} > 0$ if $e_{nm} \in \mathcal{E}$, and $a_{nm} = 0$ if $e_{nm} \notin \mathcal{E}$. Alternatively, the same graph can be described by its Laplacian matrix $L \in \mathbb{R}^{N \times N}$ defined as $L = D - A$, where $D \in \mathbb{R}^{N \times N}$ is the diagonal degree matrix such that $d_{nn} = \sum_{m} a_{nm}$. In addition, both the adjacency matrix and the Laplacian matrix can be presented in normalized form as $\hat{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ and $\hat{L} = D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$ respectively \([2]\).

In addition to capturing the underlying graph structure, both the Laplacian matrix and the adjacency matrix can be used as shift operators to filter signals on graphs. Practically, a graph shift based filter results in a linear combination of vertex-shifted graph signals, which captures graph information at a local level \([7]\). For instance, the operation $g = (1 +A)f$ results in a filtered signal $g \in \mathbb{R}^{N}$ such that:

$$g_{n} = f_{n} + \sum_{m \in \Omega_{n}} a_{nm}f_{m} \quad (4)$$

where $\Omega_{n}$ denotes the 1-hop neighbours that are directly connected to the $n$-th node. For $M$ graph signals stacked in matrix form as $F \in \mathbb{R}^{N \times M}$, equation \([4]\) can be extended as:

$$G = (1 + A)F \quad (5)$$

For reaching neighbours that are $K$-hops away, equation \([4]\) can be extended to its polynomial form as $g = \sum_{k=0}^{K} h_{k}A^{k}f$, where $h_{k}$ are constants \([7]\). Fundamentally, a $k$-hop based graph filter acts locally in the vertex space of a graph, which takes into account the irregular domain underlying the data described by its adjacency matrix.

### 2.3. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) \([11]\) are among the most successful deep learning tools for sequence modelling. A standard RNN layer captures time-varying dependencies by processing hidden states $h_{t} \in \mathbb{R}^{M}$ at time $t$ through feedback or recurrent weights as:

$$h_{t} = \sigma_{h}(W^{(h)}h_{t-1} + W^{(x)}x_{t} + b^{(h)}) \quad (6)$$

where $h_{t-1} \in \mathbb{R}^{M}$ is the hidden state vector from the previous time-step, $x_{t} \in \mathbb{R}^{N}$ is the input features vector at time $t$, $W^{(h)} \in \mathbb{R}^{M \times M}$ is the feedback weight matrix, $W^{(x)} \in \mathbb{R}^{M \times N}$ is the input weight matrix, $b^{(h)} \in \mathbb{R}^{M}$ is a bias vector, and $\sigma_{h}(\cdot)$ is an element-wise activation function.

Finally, after extracting the hidden states, these can be passed through additional weight matrices to generate outputs $y_{t} \in \mathbb{R}^{P}$ at time $t$ as:

$$y_{t} = \sigma_{y}(W^{(y)}h_{t} + b^{(y)}) \quad (7)$$

where $W^{(y)} \in \mathbb{R}^{P \times M}$ is the output weight matrix, $h_{t}$ is the hidden state at time $t$, $b^{(y)}$ is a bias vector, and $\sigma_{y}(\cdot)$ is an element-wise activation function.

### 3. Recurrent Graph Tensor Networks

#### 3.1. Special Recurrent Graph Filter

In this section, we derive the implicit graph filter underlying RNNs by considering a special case of the hidden state equation for modelling sequential data.

Consider a linear form of equation \([6]\) without bias (non-linearity and bias can be introduced later on, as discussed in section \([3.3]\)). Let $\hat{x}_{t} = W^{(x)}x_{t} \in \mathbb{R}^{M}$, then equation \([6]\) can be written in block-matrix form for $\tau$ successive time-steps as:

$$\begin{bmatrix} h_{\tau} \\ h_{\tau-1} \\ \vdots \\ h_{1} \end{bmatrix} = \begin{bmatrix} (W^{(h)})^{0} & (W^{(h)})^{1} & \cdots & (W^{(h)})^{\tau-1} \\ 0 & (W^{(h)})^{0} & \cdots & (W^{(h)})^{\tau-2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & (W^{(h)})^{0} \end{bmatrix} \begin{bmatrix} \hat{x}_{\tau} \\ \hat{x}_{\tau-1} \\ \vdots \\ \hat{x}_{1} \end{bmatrix} \quad (8)$$

We now define (i) $\hat{X} \in \mathbb{R}^{\tau \times M}$ as the matrix generated by stacking $\hat{x}_{t}$ as row-vectors over $\tau$ successive time-steps, (ii) $H \in \mathbb{R}^{\tau \times M}$ as the matrix generated by stacking hidden state vectors $h_{t}$ as row-vectors at corresponding time-steps, and (iii) $R \in \mathbb{R}^{\tau M \times \tau M}$ as the block matrix composed by the powers of $W^{(h)}$ from equation \([8]\) This allows equation \([8]\) to be expressed as:

$$\text{vec}(H^{T}) = R \times_{2} \text{vec}(\hat{X}^{T}) \quad (9)$$

Consider the special case of $W^{(h)} = cI^{(M)}$ where $c$ is a positive constant less than 1, and $I^{(M)} \in \mathbb{R}^{M \times M}$ is the identity matrix. This allows the hidden state equation to be expressed as $h_{t} = ch_{t-1} + \hat{x}_{t}$, which represents a system where the past information is propagated to the future via a damping factor $c$. For this special case, we can simplify equation \([8]\) as:

$$H = \begin{bmatrix} 1 & c^{1} & c^{2} & \cdots & c^{\tau-1} \\ 0 & 1 & c^{1} & \cdots & c^{\tau-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & c^{\tau-3} \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \hat{X} \quad (10)$$

Let $G \in \mathbb{R}^{\tau \times \tau}$ denote the upper-triangular matrix in equation \([10]\) then the same equation can be expressed in matrix form as $H = G \times_{2} \hat{X}$. However, matrix $G$ can be further decomposed as $G = I^{(\tau)} + A$, where $I^{(\tau)} \in \mathbb{R}^{\tau \times \tau}$ is the identity matrix and $A \in \mathbb{R}^{\tau \times \tau}$ is the matrix composed by the $p$-th powers of the constant $c$. Therefore, the special case of recurrent modelling can be expressed as:

$$H = (I^{(\tau)} + A) \times_{2} \hat{X} \quad (11)$$

which is a form of localized graph filter as discussed in equation \([5]\). For instance, each of the $\tau$ time-steps can be considered as a node of a graph where signals are sampled from, and $A$ is the corresponding weighted graph adjacency matrix connecting different time-steps. This also justifies the triangular nature of the graph filter, since only past information can influence future states but not vice-versa.
3.2. General Recurrent Graph Filter

In this section, we relax the restrictions from section 3.1 and extend the graph filter in equation 11 to the general case of sequence modelling.

Let the feedback matrix \( W^{(h)} \) be a scaled idempotent matrix, that is \( W^{(h)} = cW^{(r)} \) where \( c \) is a positive constant less than 1 that models the damping effect, and \( W^{(r)} \) is an idempotent matrix that models how information propagates between successive time-steps. For this setup, the feedback matrix has the property that \( (W^{(h)})^p = c^pW^{(r)} \) for \( p \) greater than 0, which allows the block matrix \( R \) to be simplified as:

\[
R = \begin{bmatrix}
I^{(M)} & cW^{(r)} & c^2W^{(r)} & \cdots & c^{r-1}W^{(r)} \\
0 & I^{(M)} & cW^{(r)} & \cdots & c^{r-2}W^{(r)} \\
0 & 0 & I^{(M)} & \cdots & c^{r-3}W^{(r)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & I^{(M)}
\end{bmatrix}
\]

(12)

The block matrix \( R \) with above properties can be further decomposed by using (i) the weighted graph adjacency matrix \( A \) from equation 11 (ii) the idempotent matrix \( W^{(r)} \), and (iii) the identity matrix \( I^{(M)} \) to be simplified as:

\[
\text{vec}(H^T) = (I^{(rM)} + (A \otimes W^{(r)})) \times \frac{1}{2} \text{vec}(X^T)
\]

(14)

Finally, we define the multi-linear graph filter \( \mathcal{R} \in \mathbb{R}^{r \times M \times r \times M} \) to be the 4-th order tensorization of the block matrix \( R \in \mathbb{R}^{r \times M \times r \times M} \) from equation 13. This allows us to simplify equation 14 without the vectorization operator via a double tensor contraction as:

\[
H = \mathcal{R} \times_{3,4}^{1,2} \hat{X}
\]

(15)

Therefore, the graph structure discussed in section 3.1 can be carried over to the general case of sequence modelling, resulting in a multi-linear graph filter \( \mathcal{R} \) capable of modelling sequential information defined on a time-graph domain.

3.3. Tensor Network Formulation

In this section, we introduce a number of Recurrent Graph Tensor Network (RGTN) models by appealing to the expressive power of tensor networks and the graph filters derived in the previous sections.

Consider the special case of graph filtering in section 3.1 where the hidden states are extracted through a graph filter that is localized in the vertex space. The extracted hidden states can be considered as feature maps, which can be flattened and passed through additional dense layers to generate application dependent outputs 13. Using the tensor network notation, we can represent the special graph filter contraction and the dense layer matrix contraction in a unique tensor network, as shown in figure 3. We will refer to this special graph filter based tensor network as the special Recurrent Graph Tensor Network (sRGTN).

Similarly, for the general case of graph filtering in section 3.2 we can represent the double contraction in equation 15 to derive the tensor network in figure 4. This is referred to as the general Recurrent Graph Tensor Network (gRGTN). Unlike the sRGTN in figure 3 where the graph filter and the feature map contractions can be modelled separately, gRGTN implies a stronger coupling of the features with the underlying graph domain, as captured by the multi-linear graph filter \( \mathcal{R} \).
For multi-modal problems, we propose a highly efficient variant of the sRGTN by appealing to the compression power of the TTD. Specifically, by reshaping dense layer matrices as higher order tensors and representing them in TT form, we can drastically reduce the parameters complexity, as discussed in [2]. This allows us to simultaneously (i) maintain the inherent tensor structure of the problem, (ii) drastically reduce the parameters complexity of the model, and (iii) incorporate the underlying graph topology. This leads to the Multi-Modal, Tensor-Train variant of sRGTN (sRGTN-TT), which is illustrated in figure 5.

Finally, we can introduce non-linearity in a given model by applying a point-wise activation function on top of a contraction. Non-linear layers can also be stacked one after another to increase the overall expressive power.

4. EXPERIMENTS

4.1. Experiment Setting

In this section, the proposed Recurrent Graph Tensor Network (RGTN) models are implemented and compared to a standard Recurrent Neural Network (RNN) to validate the proposed models for the task of time-series forecasting.

The learning task of this experiment is to forecast the PM2.5 level across multiple sites in China, using the Beijing Multi-Site Air-Quality dataset [14]. Specifically, the data consists of hourly air quality measurements of 12 variables (PM2.5, PM10, SO2, NO2, CO, O3, TEMP, PRES, DEWP, RAIN, wd, WSPM) between 2013 and 2017 across 12 different geographical sites.

The given dataset is pre-processed by (i) filling the missing data points with the corresponding feature median, (ii) scaling the numerical features between 0 and 1, and (iii) encoding the categorical features via one-hot-encoding, which increases the number of total features to 27 per site.

For the given task, a total of 4 models are implemented and compared. To achieve comparable results, all models are given the same architectural specification as shown in table 2 with the only difference in the choice of features extraction method. Specifically, the 4 features extraction methods are based on: (i) a simple recurrent neural network (RNN), (ii) a special Recurrent Graph Tensor Network (sRGTN), which implements the architecture in figure 3 (iii) a general Recurrent Graph Tensor Network (gRGTN), which implements the architecture in figure 4 and (iv) a sRGTN with TT decomposition (sRGTN-TT), which implements the architecture in figure 5. Finally, all the models are trained using the same setting, that is using (i) a stochastic gradient descent optimizer with a learning rate of $10^{-4}$, (ii) a mean-squared-error loss function, (iii) a batch size of 32, and (iv) a total of 100 epochs. Finally, the first 70% of the data is used for training purposes (20% of which is used for validation), and the remaining 30% for testing.

| RNN Model | Layer 1 | Layer 2 | Layer 3 |
|-----------|---------|---------|---------|
| Layer Type | RNN     | Dense   | Dense   |
| Units     | 8       | 12      | 12      |
| Activation | tanh    | tanh    | linear  |
| sRGTN Model | Layer 1 | Layer 2 | Layer 3 |
| Layer Type | sRGTN   | Dense   | Dense   |
| Units     | 8       | 12      | 12      |
| Activation | tanh    | tanh    | linear  |
| gRGTN Model | Layer 1 | Layer 2 | Layer 3 |
| Layer Type | gRGTN   | Dense   | Dense   |
| Units     | 8       | 12      | 12      |
| Activation | tanh    | tanh    | linear  |
| sRGTN-TT Model | Layer 1 | Layer 2 | Layer 3 |
| Layer Type | sRGTN   | TT-Dense| Dense   |
| Units     | 8       | 12      | 12      |
| Activation | tanh    | tanh    | linear  |
| TT-Rank   | n.a.    | (1,2,2,1)| n.a.    |

Table 2. Architecture of the models used in the experiment.

Note that for the sRGTN-TT model, each input data sample $X \in \mathbb{R}^{6 \times 12 \times 27}$ is kept in its natural multi-modal form, which contains 6 consecutive time-steps of data across 12 different sites, where each site contains 27 features. For RNN, sRGTN, and gRGTN models instead, each of the input data sample $X \in \mathbb{R}^{6 \times 324}$ is a matrix created by concatenating 27 features per site across all 12 sites (324 features in total) over 6 consecutive time-steps. For all models, the target prediction variable $y \in \mathbb{R}^{12}$ is the PM2.5 measurement of the successive time-step across all 12 sites.

4.2. Experiment Results

| N. Param. | RNN | sRGTN | gRGTN | sRGTN-TT |
|-----------|-----|-------|-------|----------|
| Test MSE  | 0.010935 | 0.009998 | 0.009183 | 0.008467 |

Table 3. The number of trainable parameters and the Test MSE for all proposed models. sRGTN-TT obtained the best results at a drastically lower number of parameters.

This section analyses the results obtained from the proposed experiment, demonstrating the superior properties of the proposed RGTN models over a classical RNN model, in terms of convergence, performance, and complexity properties, especially in the multi-modal setting.

Table 4.2 shows the number of trainable parameters and the final test Mean-Squared-Error (MSE) achieved by the 4 models. Results confirm the superiority of the proposed RGTN models over a standard RNN, especially in the multi-modal setting. In particular, the proposed sRGTN-TT model successfully captured the inherent multi-modality of the underlying problem, achieving the best MSE score while using up to 90% less parameters than other models.
Figure 6 shows the validation MSE of all 4 models during the training phase (in log scale). The error curves show that all RGTN models exhibited better convergence properties than the standard RNN model, as less epochs are needed to converge to the final error value.

Fig. 6. Validation MSE during 100 epochs of training for all models. RGTN models tend to converge faster and perform better than the RNN.

5. CONCLUSION

In this work, we proposed a novel Recurrent Graph Tensor Network (RGTN) by appealing to the expressive power of tensor networks and graph signal processing methods. We provided the theoretical framework underpinning the proposed RGTN models and applied them to the task of time-series forecasting. Our experiment results show that the proposed RGTN models exhibit desirable properties in terms of convergence, performance, and complexity. Specifically, when dealing with multi-modal data, the proposed RGTN models out-performed a standard RNN both in terms of performance (45% improvement in mean-squared-error) and complexity (90% reduction in trainable parameters).

6. REFERENCES

[1] Danilo P Mandic and Jonathon Chambers, Recurrent neural networks for prediction: learning algorithms, architectures and stability, John Wiley & Sons, Inc., 2001.

[2] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov, “Tensorizing neural networks,” in Proceedings of the Advances in Neural Information Processing Systems (NIPS), 2015, pp. 442–450.

[3] Nadav Cohen, Or Sharir, and Amnon Shashua, “On the expressive power of deep learning: A tensor analysis,” in Conference on Learning Theory, 2016, pp. 698–728.

[4] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip, “A comprehensive survey on graph neural networks,” IEEE Transactions on Neural Networks and Learning Systems, 2020.

[5] A. Cichocki, “Era of big data processing: A new approach via tensor networks and tensor decompositions,” ArXiv e-prints, Mar. 2014.

[6] Ivan V Oseledets, “Tensor-train decomposition,” SIAM Journal on Scientific Computing, vol. 33, no. 5, pp. 2295–2317, 2011.

[7] Ljubisa Stankovic, Danilo Mandic, Milos Dakovic, Milos Brajovic, Bruno Scalzo, and Anthony G Constantinides, “Graph signal processing–part ii: Processing and analyzing signals on graphs,” arXiv preprint arXiv:1909.10325, 2019.

[8] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst, “Geometric deep learning: going beyond euclidean data,” IEEE Signal Processing Magazine, vol. 34, no. 4, pp. 18–42, 2017.

[9] S.V. Dolgov and D.V. Savostyanov, “Alternating minimal energy methods for linear systems in higher dimensions,” SIAM Journal on Scientific Computing, vol. 36, no. 5, pp. A2248–A2271, 2014.

[10] Andrzej Cichocki, Namgil Lee, Ivan Oseledets, Anh-Huy Phan, Qibin Zhao, Danilo P Mandic, et al., “Tensor networks for dimensionality reduction and large-scale optimization: Part I low-rank tensor decompositions,” Foundations and Trends® in Machine Learning, vol. 9, no. 4-5, pp. 249–429, 2016.

[11] Ivan V Oseledets and Eugene E Tyrtyshnikov, “Breaking the curse of dimensionality, or how to use SVD in many dimensions,” SIAM Journal on Scientific Computing, vol. 31, no. 5, pp. 3744–3759, 2009.

[12] Ljubisa Stankovic, Danilo Mandic, Milos Dakovic, Milos Brajovic, Bruno Scalzo, and Tony Constantinides, “Graph signal processing–part i: Graphs, graph spectra, and spectral clustering,” arXiv preprint arXiv:1907.03467, 2019.

[13] Md Zahangir Alom, Tarek M Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C Van Essen, Abdul AS Awwal, and Vijayan K Asari, “A state-of-the-art survey on deep learning theory and architectures,” Electronics, vol. 8, no. 3, pp. 292, 2019.

[14] Shuyi Zhang, Bin Guo, Anlan Dong, Jing He, Ziping Xu, and Song Xi Chen, “Cautionary tales on air-quality improvement in beijing,” Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 473, no. 2205, pp. 20170457, 2017.