Tensor-Variate Machine Learning on Graphs

Yao Lei Xu

Communications & Signal Processing Group
Department of Electrical & Electronic Engineering
Imperial College London

A thesis submitted for the degree of
Doctor of Philosophy
2023

Abstract

Traditional machine learning algorithms are facing significant challenges as the world enters the era of big data, with a dramatic expansion in volume and range of applications and an increase in the variety of data sources. The large- and multi-dimensional nature of data often increases the computational costs associated with their processing and raises the risks of model over-fitting - a phenomenon known as the *curse of dimensionality*. To this end, *tensors* have become a subject of great interest in the data analytics community, owing to their remarkable ability to super-compress high-dimensional data into a low-rank format, while retaining the original data structure and interpretability. This leads to a significant reduction in computational costs, from an exponential complexity to a linear one in the data dimensions.

An additional challenge when processing modern big data is that they often reside on irregular domains and exhibit relational structures, which violates the regular grid assumptions of traditional machine learning models. To this end, there has been an increasing amount of research in generalizing traditional learning algorithms to *graph* data. This allows for the processing of graph signals while accounting for the underlying relational structure, such as user interactions in social networks, vehicle flows in traffic networks, transactions in supply chains, chemical bonds in proteins, and trading data in financial networks, to name a few.

Although promising results have been achieved in these fields, there is a void in literature when it comes to the conjoint treatment of tensors and graphs for data analytics. Solutions in this area are increasingly urgent, as modern big data is both large-dimensional and irregular in structure. To this end, the goal of this thesis is to explore machine learning methods that can fully exploit the advantages of both tensors and graphs. In particular, the following approaches are introduced: (i) Graph-regularized tensor regression framework for modelling high-dimensional data while accounting for the underlying graph structure; (ii) Tensor-algebraic approach for computing efficient convolution on graphs; (iii) Graph tensor network framework for designing neural learning systems which is both general enough to describe most existing neural network architectures and flexible enough to model large-dimensional data on any and many irregular domains. The considered frameworks were employed in several real-world applications, including air quality forecasting, protein classification, and financial modelling. Experimental results validate the advantages of the proposed methods, which achieved better or comparable performance against state-of-the-art models. Additionally, these methods benefit from increased interpretability and reduced computational costs, which are crucial for tackling the challenges posed by the era of big data.

Contents

List of Figures **1**

List of Tables **11**

List of Proposed Algorithms **13**

Nomenclature **14**

Statement of Originality **17**

Copyright Declaration **18**

Acknowledgments **19**

Publications **20**

1 Introduction **23**
1.1 Challenges of the Modern Big Data 23
1.2 A Brief History of Tensors and Graphs 27
1.3 Motivation and Aims of the Thesis 28
1.4 Thesis Organization 29
1.5 Contributions 30

2 Preliminaries **37**
2.1 Preliminaries on Tensors 37
2.2 Preliminaries on Graphs 46

I Tensor Learning Methods **50**

3 Tensor Contraction via Tensor-Train Decomposition **51**
3.1 Introduction 51
3.2 Contraction in the Tensor-Train Format 53
3.3 Computational Complexity Analysis 56
3.4 Numerical Examples 58
3.5 Approximation Error 59
3.6 Summary 59

4 Tensor-Train Recurrent Neural Networks for Finance 61
4.1 Introduction . . . 61
4.2 Preliminaries . . . 62
4.2.1 *Tensor-Train Fully Connected Neural Networks* . . . 62
4.2.2 *Tensor-Train Recurrent Neural Networks* . . . 63
4.3 Tensor Train Recurrent Neural Network for Finance . . . 63
4.3.1 *Input Data Structure* . . . 63
4.3.2 *Model Training Settings* . . . 65
4.3.3 *Experimental Results* . . . 66
4.3.4 *Interpreting Relative Modal Importance* . . . 67
4.4 Summary . . . 68

II Graph Learning Methods 69

5 Graph Theory for Metro Traffic Modelling 70
5.1 Introduction . . . 70
5.2 Traffic Centrality and Vulnerability . . . 72
5.3 K-Edge Augmentation for Robust Traffic Networks . . . 73
5.4 Modelling Net Passenger Flow and Population . . . 75
5.5 A Hypergraph Model of Metro Network . . . 80
5.6 Summary . . . 82

6 Hierarchical Graph Pooling for Finance 83
6.1 Introduction . . . 83
6.2 Preliminaries . . . 85
6.2.1 *Graph Representation of the Stock Market* . . . 85
6.2.2 *Graph Convolutional Network* . . . 85
6.2.3 *Select-Reduce-Connect Framework for Graph Pooling* . . . 85
6.2.4 *Graph Neural Network Architecture* . . . 86
6.3 The Stock Pooling Operator . . . 86
6.4 Graph Ensemble Learning . . . 87
6.5 Simulations . . . 88
6.6 Summary . . . 89

III Tensor-Variate Machine Learning on Graphs 90

7 Graph-Regularized Tensor Regression 91
7.1 Introduction . . . 91
7.2 Related Work . . . 93
7.3 Graph-Regularized Tensor Regression . . . 93
7.3.1 *Tensor Regression* . . . 93
7.3.2 *Graph-Regularized Tensor Regression* . . . 95
7.3.3 *Choice of Model Rank* . . . 98
7.3.4 *Choice of Regularization Parameter* . . . 99

7.3.5 *Computational Complexity Analysis* 99
7.4 Experiments on Synthetic Data 101
7.4.1 *Experimental Setting* 101
7.4.2 *Data* 101
7.4.3 *Incorporation of Domain Knowledge with Graphs* 102
7.4.4 *Models* 102
7.4.5 *Results* 103
7.5 Experiments on Real World Data 103
7.5.1 *Data* 103
7.5.2 *Incorporation of Domain Knowledge with Graphs* 104
7.5.3 *Models* 104
7.5.4 *Results* 105
7.6 Interpretability 105
7.6.1 *Coefficient-Wise Interpretability* 105
7.6.2 *Mode-Wise Interpretability* 105
7.7 Summary 108

8 Graph Convolution via Convolution Tensors 109
8.1 Introduction 109
8.2 Related Work 110
8.3 Preliminaries 111
8.3.1 *Convolution and Convolution Tensor* 111
8.4 Convolution Tensor for Graphs 111
8.4.1 *Circulant Graph Nature of the Convolution Tensor* 111
8.4.2 *Convolution Tensor for 2D Convolution* 114
8.4.3 *General Graph Convolution Tensors* 116
8.4.4 *Interpretation of Graph Convolution via Graph Convolution Tensor* 118
8.4.5 *Efficient and Scalable Graph Convolution* 119
8.4.6 *Multi-Channel Graph Convolution* 119
8.4.7 *Graph Convolution Tensor Network* 120
8.5 Experiments 120
8.6 Summary 121

9 Recurrent Graph Tensor Networks 122
9.1 Introduction 122
9.2 Preliminaries 123
9.2.1 *Recurrent Neural Networks* 123
9.3 Recurrent Graph Tensor Networks 124
9.3.1 *General Recurrent Graph Tensor Networks* 124
9.3.2 *Simplified Recurrent Graph Tensor Networks* 125
9.3.3 *Tensor Network Formulation* 125
9.4 Experiments 127
9.4.1 *Datasets* 127
9.4.2 *Benchmark Models and Metrics* 128
9.4.3 *Experiment Results* 128
9.5 Further Experiments on Financial Data 129

9.5.1 *Preliminaries on Limit Order Books* . . . 129
9.5.2 *Experiment Overview* . . . 130
9.5.3 *Dataset* . . . 130
9.5.4 *Benchmark Models and Metrics* . . . 131
9.5.5 *Experiment Results* . . . 132
9.6 Summary . . . 133

10 Attention Graph Tensor Networks **136**
10.1 Introduction . . . 136
10.2 Preliminaries . . . 138
10.2.1 *Dot Product Attention* . . . 138
10.2.2 *Graph Tensor Networks* . . . 138
10.3 Spectral Attention . . . 139
10.3.1 *Time-Domain Attention Graph* . . . 139
10.3.2 *Spectral Attention Filtering* . . . 140
10.4 Tensorizing Attention . . . 141
10.4.1 *Tensorizing Input Data* . . . 141
10.4.2 *Tensorizing Large Dimensional Matrix Multiplications* . . . 141
10.4.3 *Tensorizing Attention Coefficients* . . . 142
10.5 Putting Everything Together . . . 142
10.6 Experimental Results . . . 144
10.7 Summary . . . 146

11 Generalized Graph Tensor Networks **147**
11.1 Introduction . . . 147
11.2 Related Work . . . 148
11.3 Preliminaries . . . 149
11.4 Multi-Domain Data Tensors: . . . 149
11.5 The Graph Tensor Network Framework . . . 150
11.5.1 *The Forward Pass* . . . 151
11.6 Classical Neural Networks as GTNs . . . 152
11.6.1 *Dense Neural Network* . . . 152
11.6.2 *Graph Convolutional Network* . . . 152
11.6.3 *Convolutional Neural Network* . . . 153
11.6.4 *Attention Neural Network* . . . 154
11.6.5 *Recurrent Neural Network* . . . 155
11.7 Beyond Classical Neural Networks . . . 157
11.8 Experiments . . . 157
11.9 Summary . . . 160

12 Conclusions and Future Work **161**
12.1 Research Achievements and Future Work . . . 161

Bibliography **166**

List of Figures

1.1 The 4 Vs of the modern big data [17]: (i) Volume - scale of data; (ii) Variety - different forms of data; (iii) Veracity - uncertainty of data; (iv) Velocity - analysis of streaming data. 24

1.2 Tensors as a generalization of scalars, vectors, and matrices. 25

1.3 Order-4 tensor representation of the financial big data. Market data features (e.g., price) can be collected over time (e.g., daily) for each component (e.g., the Microsoft stock) of every asset class (e.g., equities). 25

1.4 Converting temperature measurements from multiple sensors into a graph signal (adapted from [34]). The top-left panel illustrates the location of temperature measurements. The bottom-left panel illustrates the classical vector representation of the temperature signal, which is unaware of the underlying spatial structure. The top-right panel represents the top-left panel in a graph form, where the vertices represent the sensor locations while the edges denote their physical proximity. The colour of the vertices is proportional to the temperature at the associated location. The bottom-right panel illustrates the graph signal representation of temperature measurements, which is aware of the underlying geographical structure. 26

1.5 Multi-graph representation of a social network across 4 modes of communication (adapted from [36]). This can be modelled via a tensor, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{3\times3\times4}$, where (i_1, i_2, i_3)-th entry denote whether the i_1-th person is connected to the i_2-th person via the i_3-th mode of communication. 28

1.6 Thesis Structure. 36

2.1 Matricization of an order-3 tensor with respect to mode-1. 38

2.2 Vectorization of an order-2 tensor. 38

2.3 Tensor (top) and matricized (bottom) view of a mode-$n = 1$ product between an order-3 tensor, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1\times I_2\times I_3}$, and matrix, $\boldsymbol{B} \in \mathbb{R}^{J\times I_1}$, to yield, $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{J\times I_2\times I_3}$. 40

2.4 TN diagrams for basic tensor operations: (1) Matrix transpose from $\boldsymbol{A} \in \mathbb{R}^{I_1 \times I_2}$ to $\boldsymbol{A}^T \in \mathbb{R}^{I_2 \times I_1}$; (2) Matrix vectorization from $\boldsymbol{A} \in \mathbb{R}^{I_1 \times I_2}$ to $\bar{\boldsymbol{a}} \in \mathbb{R}^{I_1 I_2}$; (3) Tensor matricization from $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ to $\boldsymbol{A}_{\{1\}} \in \mathbb{R}^{I_1 \times I_2 I_3}$; (4) Tensor contraction, $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}} \times_1^4 \boldsymbol{\mathcal{B}}$, with $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{J_1 \times J_2 \times J_3 \times J_4}$ to yield $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{I_2 \times I_3 \times J_1 \times J_2 \times J_3}$; (5) Mode-$n$ product, $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}} \times_2^2 \boldsymbol{B}$, with $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and $\boldsymbol{B} \in \mathbb{R}^{I_1 \times J}$ to yield $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{I_2 \times I_3 \times J}$; (6) Tucker product, $\boldsymbol{\mathcal{C}} = [\![\boldsymbol{\mathcal{A}}; \boldsymbol{B}^{(1)}, \boldsymbol{B}^{(2)}, \boldsymbol{B}^{(3)}, \boldsymbol{B}^{(4)}]\!]$, with $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4}$ and $\boldsymbol{B}^{(n)} \in \mathbb{R}^{I_n \times J_n}$ for $n = 1, \ldots, 4$ to yield $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{J_1 \times J_2 \times J_3 \times J_4}$; (7) TTD, $\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{G}}^{(1)} \times_4^1 \boldsymbol{\mathcal{G}}^{(2)} \times_4^1 \boldsymbol{\mathcal{G}}^{(3)} \times_4^1 \boldsymbol{\mathcal{G}}^{(4)}$, with $\boldsymbol{\mathcal{G}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times J_n \times R_n}$ for $n = 1, \ldots, 4$ to yield $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4}$. . . 41

2.5 A TT type TN decomposition of an order-5 tensor, $\boldsymbol{\mathcal{X}}$, into three order-3 small-scale cores, $\boldsymbol{\mathcal{G}}^{(2)}, \boldsymbol{\mathcal{G}}^{(3)}, \boldsymbol{\mathcal{G}}^{(4)}$, and two order-2 factor matrices, $\boldsymbol{A} = \boldsymbol{\mathcal{G}}^{(1)}$ and $\boldsymbol{B} = \boldsymbol{\mathcal{G}}^{(5)}$. 43

2.6 Graphical representation of TTD of an order-5 tensor, $\boldsymbol{\mathcal{X}}$ in TN notation. 43

2.7 Visualization of the CPD for an order-3 tensor, $\boldsymbol{\mathcal{W}}$, according to (2.8). The sizes of tensor modes, I_1, I_2, and I_3, are labelled in gray. 44

2.8 Alternative visualization of the CPD for an order-3 tensor, $\boldsymbol{\mathcal{W}}$, according to (2.9) in standard notation. The sizes of tensor modes, I_1, I_2, and I_3, are labelled in gray. 45

2.9 Alternative visualization of the CPD for an order-3 tensor, $\boldsymbol{\mathcal{W}}$, according to (2.9) in TN notation. The sizes of tensor modes, I_1, I_2, and I_3, are labelled in gray. 45

2.10 The graph adjacency matrix (top panels) and the corresponding graph topology (bottoms) for (i) a directed graph, (ii) an undirected graph, (iii) a K-regular graph with $K = 4$, and (iv) a DCG with offset $k = 1$. 47

2.11 Smooth (left) and non-smooth (right) graph signals. The vertices of the illustrated graphs are coloured proportionally to the graph signal values, while the edges are also coloured proportionally to their edge weight. A graph signal is considered smooth if it varies slowly over closely connected neighbouring vertices. 48

3.1 Graphical illustration of the TTCP. The contraction is performed on an order-4 tensor, $\boldsymbol{\mathcal{X}}$, and an order-3 tensor, $\boldsymbol{\mathcal{Y}}$, along their common modes $I_3 = J_2$. By exploiting the topology of the TT representations of the original tensors, $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$, the TT decompositions of the permuted versions of $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$ are obtained, allowing for the reduction of the otherwise computationally prohibitive contraction in (2.5) to a single matrix multiplication. 55

3.2 Theoretical time complexity (in log-scale) required to perform the proposed TTCP for tensors in TT-format with varying dimensions, I, and TT ranks, R. 57

3.3 Theoretical time complexity (in log-scale) required to perform the standard contraction product for tensors with varying dimensions, I, and tensor orders, N. 57

3.4 Computational time (in seconds, log-scale) of the TCP and the proposed TTCP, for order-3, order-4 and order-5 tensors. 58

3.5 Theoretical time complexity (in number of operations, log-scale) of the TCP and the proposed TTCP corresponding to Figure 3.4. 58

3.6 The TTCP approximation error measured in explained variance (y-axis) as a function of TT rank (x-axis) across three popular ML datasets. Note that a TTCP with a low rank of $R_n = 4, n = 1, \ldots, N-1$ is often enough to approximate the original contraction product by more than 90%. 59

4.1 Financial data represented as a multi-modal tensor. 64

4.2 Modal connections between every mode of the input data tensor, $\boldsymbol{\mathcal{X}}_t$, and the TT-cores, $\boldsymbol{\mathcal{G}}_n$, of the tensorized weight matrix $\boldsymbol{W}^{xh}$ in TT-format. 66

4.3 Cumulative profits generated by the models. 67

4.4 Normalized change of TT-core weights in TTRNN. 68

5.1 Graph representation of the London underground network in Zones 1 and 2. The circles denote the vertices (stations) and the lines between the circles designate the underground lines. 71

5.2 Betweenness centrality, designated by magenta-coloured bars, of the London underground network in Zones 1 and 2. The largest betweenness centrality is observed for at stations with multiple metro lines parsing through. 73

5.3 Graph of the London underground network in Zones 1 and 2, after performing a naive k-edge augmentation, for k=2, to improve the robustness of the network. The new lines are designated in magenta, while k=2 means that no station can be disconnected from the rest of the network if only one underground line passing through that station is not operational. 74

5.4 Graph of the London underground network in Zones 1 and 2 after performing geographically constrained k-edge augmentation, for k=2, to improve the robustness of the network. Observe that the new segments, designated in magenta, are of similar length as the already existing segments. 74

5.5 A graph representation of the London underground network through Fick's law of diffusion. Consider a simplified path graph with two stations surrounded by the respective populations, ϕ_1 and ϕ_2 (proportional to the circle area), which exhibit the corresponding net fluxes, q_1 and q_2. Stations surrounded by large population (residential areas) exhibit a net in-flow of passengers, while stations surrounded by low population (business districts) experience net out-flow of passengers. The overall net flow (in-flow and out-flow) of passengers across the entire network sums up to zero. 75

5.6 Net passenger out-flow during the morning rush hour within the Zones 1 and 2 of the London underground network (for clarity, data for only several stations in residential areas are shown). The magenta bars denote a net out-flow of passengers while the blue bars designate a net in-flow of passengers. Stations located within business districts exhibit the greatest net out-flow of passengers, while stations located in residential areas, toward the outskirts, exhibit the largest net in-flow of passengers. 77

5.7 Population density implied by our graph model in (5.4), obtained from the net passenger out-flow during the morning rush hour within Zones 1 and 2. As expected, business districts exhibit the lowest population density, while residential areas (Zone 2) exhibit the highest commuter population density. 78

5.8 Ground-truth population (top) vs diffusion implied population (bottom) using synthetic data corrupted by Gaussian noise with 0.5 standard deviation. 79

5.9 A graph (left) is a special instance of a hypergraph (right), whereby each hyperedge contains exactly two nodes. A hypergraph, which can be described by its incidence matrix, $\boldsymbol{M}$, captures high-order connections among vertices, as there is no restriction on the number of nodes per hyperedge, or the number of edges or hyperedges per vertex. For example, the hyperedge e_2 connects nodes 2, 3, and 4, as also evidenced in the corresponding incidence matrix $\boldsymbol{M}$. 80

5.10 The evening net passenger-flow, estimated from the morning rush hour data, based on the hypergaph neural network model. The power of the proposed hypergraph model is reflected in the reverse net flow from the morning rush hours, as people move from central business areas back to residential zones. 82

6.1 Illustration of Two-Layer hierarchical *StockPool*. 87

6.2 A graph ensemble architecture, comprising DiffPool, MinCUTPool, and StockPool GNN. 88

6.3 Evaluation of GNN models against S&P500 market. 89

7.1 TN diagram of the inner product, $\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}} \rangle$, within an order-3 TR. The weight tensor, $\boldsymbol{\mathcal{W}}$, on the left is factorized by the CPD according to equation (2.9). The connected tensor modes represent tensor contractions along these modes. 94

7.2 Principle of GRTR. The order-3 weight tensor, $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{T \times S \times F}$, is represented as a rank-R CPD TD, where the corresponding factor matrices are of dimensions, $\boldsymbol{U}^{(1)} \in \mathbb{R}^{T \times R}$, $\boldsymbol{U}^{(2)} \in \mathbb{R}^{S \times R}$, and $\boldsymbol{U}^{(3)} \in \mathbb{R}^{F \times R}$. Our aim is to regularize the learning of the mode-2 factor matrix, $\boldsymbol{U}^{(2)} \in \mathbb{R}^{S \times R}$, such that it reflects the physical proximity information of the S sensors as domain knowledge. To this end, we construct a graph with S vertices representing S sensors, where an edge connects a pair of vertices with an edge weight proportional to their physical proximity. Given such a mode-2 graph, the corresponding factor matrix, $\boldsymbol{U} = [\boldsymbol{u}_1^{(n)}, \ldots, \boldsymbol{u}_R^{(n)}]$, can be viewed as a set of R graph signals (its column vectors), which can be learned so as to be smooth with respect to the underlying sensors graph. This makes it possible to regularize the learning of the factor matrix and results, as desired, in similar weights for neighbouring sensors. 95

7.3 Visualization of the effect of the graph regularization parameter, $\lambda^{(n)}$, on the estimated factor vectors, $\boldsymbol{u}^{(n)} \in \mathbb{R}^{I_n}$. The row panels illustrate $N = 4$ different graph domains, each with $I_n = 10$ vertices that are coloured proportionally to the values of the corresponding factor vectors. The column panels illustrate the factor vectors for different values of $\lambda^{(n)}$. Notice how the factor vectors become smoother with respect to the underlying graph with an increase in regularization parameter, as indicated by the neighbouring vertices exhibiting increasingly similar colours with larger $\lambda^{(n)}$. 99

7.4 Space (left) and time (right) complexity simulations for a TR model of rank $R = 3$. The space complexity (i.e., number of parameters) is linear in the tensor order for the low-rank form, $\mathcal{O}(NIR)$, but exponential for the full tensor form, $\mathcal{O}(I^N)$. The time complexity (in seconds) is exponential in the tensor order for both the low-rank form, $\mathcal{O}(RI^N)$, and the full tensor form, $\mathcal{O}(I^N)$, which are asymptotically the same for large N. 100

7.5 Experimental setting of the synthetic data experiment in TN notation. The training set consists of $M = 100$ samples of order-4 input tensors, $\boldsymbol{\mathcal{X}}^{(m)} \in \mathbb{R}^{10 \times 10 \times 10 \times 10}$, and the corresponding noisy regression labels, $y^{(m)}$. The goal of the experiment is to estimate the ground-truth factor matrices, $\boldsymbol{U}^{(n)} \in \mathbb{R}^{10 \times 5}$, $n = 1, \ldots, 4$, from the training set data. . . . 102

7.6 Experimental setting of the real data experiment in TN notation. The training set consists of $M = 1460$ samples of order-3 input tensors, $\boldsymbol{\mathcal{X}}^{(m)} \in \mathbb{R}^{12 \times 12 \times 27}$, and the corresponding regression labels, $y^{(m)}$. The goal of the experiment is to estimate a GRTR model from the training set data that performs well out-of-sample. 104

7.7 Illustration of a rank $R = 1$ order $N = 3$ TR operation. The data tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{T \times S \times F}$, contracts with the factor vectors, $\boldsymbol{u}_1^{(n)}$, $n = 1, 2, 3$, to generate the overall scalar output, $y = \boldsymbol{\mathcal{X}} \times_1^1 \boldsymbol{u}_1^{(1)} \times_2^1 \boldsymbol{u}_1^{(2)} \times_3^1 \boldsymbol{u}_1^{(3)}$. Each mode-$n$ contraction represents a linear combination in the n-th subspace of $\boldsymbol{\mathcal{X}}$, which reduces the order of the tensor until a scalar is obtained. In the above example, the original order-3 tensor is reduced to a matrix through a tensor-by-vector contraction (defined in Section 2.1), which is then reduced to a vector through a matrix-by-vector multiplication, which is then reduced to a scalar through a vector-by-vector inner product. 106

7.8 Estimated factor vectors of the rank $R = 1$ GRTR model from the experiment in Section 7.5. A factor vector is estimated for each of the time, site, and feature modes, in order to predict the future `PM2.5` values at the `Huairou` site. The time-mode factor vector, $\boldsymbol{u}^{(1)} \in \mathbb{R}^T$, can be interpreted as an exponentially weighted moving average operation in the time-subspace, as larger weights are placed towards more recent data. The site-mode factor vector, $\boldsymbol{u}^{(2)} \in \mathbb{R}^S$, indicates that the 7-th site (`Huairou`) is the most useful for forecasting its own future `PM2.5` value. The feature-mode factor vector, $\boldsymbol{u}^{(3)} \in \mathbb{R}^F$, indicates that the 1-st and 2-nd features (`PM2.5` and `PM10`) are the most useful for forecasting `PM2.5` value. 107

7.9 Estimated mode-2 factor vector, $\boldsymbol{u}^{(2)} \in \mathbb{R}^S$, of the rank $R = 1$ GRTR model from the experiment in Section 7.5, illustrated as a graph signal. The graph formulation of the proposed model allows for enhanced interpretability, as the weights of the obtained model can be plotted against the corresponding graph to reveal the effect of the underlying relational structure. 107

8.1 Input signal vector, $\boldsymbol{x} \in \mathbb{R}^5$, represented as a vector (left), a directed graph signal (middle), and DCG signal (right). The nodes represent time-steps, while the edges connect the time-steps in time-increasing order with weights proportional to the time difference. 111

8.2 Standard representation of the discrete convolution. This can be written in element-wise form as $\boldsymbol{y}_j = \sum_{k=1}^{K} \boldsymbol{x}_{(j+k-1)\%I} \boldsymbol{w}_k$. Note that the indices of the element-wise variables, $(\boldsymbol{y}_j, \boldsymbol{x}_{(j+k-1)\%I}, \boldsymbol{w}_k)$, correspond exactly to the non-zero entries of the convolution tensor, $\boldsymbol{\mathcal{T}}_{j,j+k-1,k} = 1$ for $j = 1, \ldots, J$ and $k = 1, \ldots, K$. 112

8.3 Slices of the convolution tensor, $\boldsymbol{\mathcal{T}}_{j,:,:} \in \mathbb{R}^{I \times K}$ for $j = 1, \ldots, J$, where $I = J = 5$ and $K = 3$. The non-zero entries of the convolution tensor links each output entry (the j-th node), to its K nearest neighbour input entries (the neighbouring nodes, $i \in \mathcal{N}_j$), and the corresponding convolution kernel entries. For instance, for the $j = 1$ output entry, $\boldsymbol{y}_1$, its $K = 3$ nearest neighbour graphs signals are associated with the $i = 1, 2, 3$ input entries, $(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$, which are mapped to the $k = 1, 2, 3$ kernel entries, $(\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3)$. Instead, for the $j = 5$ output entry, $\boldsymbol{y}_5$, its $K = 3$ nearest neighbour graph signals are associated with the $i = 5, 1, 2$ input entries, $(\boldsymbol{x}_5, \boldsymbol{x}_1, \boldsymbol{x}_2)$, which are mapped to the $k = 1, 2, 3$ kernel entries, $(\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3)$. 112

8.4 Illustration of the adjacency matrix, $\tilde{\boldsymbol{W}}$, generated using the convolution tensor, $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{5 \times 5 \times 3}$, and the convolution kernel, $\boldsymbol{w} = [3, 2, 1] \in \mathbb{R}^3$, according to (8.3). The top figures illustrate: (i) the $k = 1$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,1} \in \mathbb{R}^{5 \times 5}$; (ii) the $k = 2$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,2} \in \mathbb{R}^{5 \times 5}$; (iii) the $k = 3$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,3} \in \mathbb{R}^{5 \times 5}$; (iv) the adjacency matrix computed as a weighted sum of convolution tensor slices, $\tilde{\boldsymbol{W}} = \sum_{k=1}^{3} \boldsymbol{w}_k \boldsymbol{\mathcal{T}}_{:,:,k} \in \mathbb{R}^{5 \times 5}$. The bottom figures illustrate the corresponding graph topology implied by the adjacency matrix in the top figures. Note that the k-th tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,k}$, models the k-th nearest neighbour connection. For instance, $k = 1$ models self-connections, $k = 2$ models connections to successive time-steps, and $k = 3$ models connections 2 time-steps away. 113

8.5 Illustration of the convolution operator, $\tilde{\boldsymbol{W}} \in \mathbb{R}^{J \times I}$, generated using the convolution tensor, $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{J \times I \times K}$, and the convolution kernel, $\boldsymbol{w} = [1, 2, 3, 4] \in \mathbb{R}^K$. The top figures illustrate: (i) the $(k_1, k_2) = (1, 1)$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,1} \in \mathbb{R}^{J \times I}$; (ii) the $(k_1, k_2) = (1, 2)$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,2} \in \mathbb{R}^{J \times I}$; (iii) the $(k_1, k_2) = (2, 1)$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,3} \in \mathbb{R}^{J \times I}$; (iv) the $(k_1, k_2) = (2, 2)$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,4} \in \mathbb{R}^{J \times I}$; (v) the convolution operator computed as a weighted sum of convolution tensor slices, $\tilde{\boldsymbol{W}} = \sum_{k=1}^{K} \boldsymbol{w}_k \boldsymbol{\mathcal{T}}_{:,:,k} \in \mathbb{R}^{J \times I}$. The bottom figures illustrate the corresponding graph topology implied by the adjacency matrix in the top figures. Note that the k-th tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,k}$, models the k-th nearest neighbour connection. 114

8.6 Illustration of the 2D convolution operator, $\tilde{\boldsymbol{\mathcal{W}}} = \left(\boldsymbol{\mathcal{T}} \times_{5,6}^{1,2} \boldsymbol{W}\right) \in \mathbb{R}^{J_1 \times J_2 \times I_1 \times I_2}$, given the convolution kernel, $\boldsymbol{W} = [[1, 2], [3, 4]] \in \mathbb{R}^{K_1 \times K_2}$, where $I_1 = I_2 = J_1 = J_2 = 5$ and $K_1 = K_2 = 2$. The (j_1, j_2) sub-figure illustrates the corresponding convolution operator slice, $\tilde{\boldsymbol{\mathcal{W}}}_{j_1, j_2, :, :} \in \mathbb{R}^{I_1 \times I_2}$. 115

8.7 Illustration of the convolution operator, $\tilde{\boldsymbol{W}} \in \mathbb{R}^{5\times5}$, generated using the convolution tensor, $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{5\times5\times3}$, and the convolution kernel, $\boldsymbol{w} = [3, 2, 1] \in \mathbb{R}^3$, associated with Example 7. The top figures illustrate: (i) the $k = 1$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,1} \in \mathbb{R}^{5\times5}$; (ii) the $k = 2$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,2} \in \mathbb{R}^{5\times5}$; (iii) the $k = 3$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,3} \in \mathbb{R}^{5\times5}$; (iv) the convolution operator computed as a weighted sum of convolution tensor slices, $\tilde{\boldsymbol{W}} = \sum_{k=1}^{3} \boldsymbol{w}_k \boldsymbol{\mathcal{T}}_{:,:,k} \in \mathbb{R}^{5\times5}$. The bottom figures illustrate the corresponding graph topology implied by the adjacency matrix in the top figures. Note that the k-th tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,k}$, models the k-th nearest neighbour connection. In contrast to the classical convolution tensor illustrated in Figure 8.4, the irregular structure of the graph is reflected in the irregular positions of non-zero entries in the convolution tensor slices, since not every node has a k-th nearest neighbour and the ranking of neighbouring nodes is not successive. 118

8.8 The structure of matrices $\tilde{\boldsymbol{W}} \in \mathbb{R}^{J\times K}$ and $\tilde{\boldsymbol{X}} \in \mathbb{R}^{J\times I}$ generated from $\boldsymbol{w} \in \mathbb{R}^3$ and $\boldsymbol{x} \in \mathbb{R}^5$. 119

9.1 TN diagram of the gRGTN filtering operation (left) according to (9.8), and the sRGTN filtering operation (right) according to (9.9). The nodes of the TN diagram represent different tensors, while the edges represent tensor contractions over common dimensions between tensors. The dimensions of different tensors are denoted in gray letters. 125

9.2 A gRGTN model designed to handle order-3 input data tensors. In addition to the proposed RGTN filtering operation, it leverages the power of TTD networks (in yellow) to achieve high expressive power at low parameter complexity, which is inherently compatible with the multi-way nature of the gRGTN. 126

9.3 An sRGTN model designed to handle order-3 input time-series tensors. Similar to gRGTN, it leverages on the power of TTD networks (in yellow) to achieve high expressive power at low parameter complexity, which is inherently compatible with the multi-way nature of the RGTN framework. 126

9.4 Tensor structure of the order book. The order book data of a stock at a given point in time contains the buy side (bid) and sell side (ask) information about pricing and the size. The l-th best bid and ask available are contained within the l-th level of the order book. By considering the order book information across L levels over T time steps, we obtain order-4 data tensors of [time] $\times$ [price and size] $\times$ [bid and ask] $\times$ [order book levels]. 129

9.5 Experiment results. (Left) MSEs and their variances: The RGTN models provide lower MSEs and smaller output variances than TTRNN for each of the five stocks considered. Both outperform significantly the GRU benchmark model. (Right) Gradients: Evolution of gradients of each model considered across training epochs. Tensorized models display better behavior, with fastest convergence of gradients in each instance of RGTN experiments. 134

10.1 Principle of the time-domain attention graph (in purple), computed from the bi-directional time-graph (in red), and the symmetric attention coefficient graph (in blue), as a Hadamard product of their adjacency matrices, $\mathbf{\Psi} = \mathbf{\Omega} \odot \mathbf{\Theta}$. The graph vertices represent a sequence of tokens embedded in the tensor space, which are interconnected with edge weights proportional to their time-distance, $\mathbf{\Omega}_{l_1,l_2}$, and attention coefficient, $\mathbf{\Theta}_{l_1,l_2}$, for $l_1 = 1, \ldots, L$ and $l_2 = 1, \ldots, L$. . . 140

10.2 Principle of the TSA mechanism. A sequence of tokens are embedded in the tensor space (in gray). The sequence of tensor embeddings are then mapped to generate the key tensors, $\boldsymbol{\mathcal{K}}$ (in orange), and the value tensors, $\boldsymbol{\mathcal{V}}$ (in green). The key tensors are then used to generate the adjacency matrix of the attention graph, $\mathbf{\Psi}$, which is used to perform spectral graph convolution on value tensors to generate the output tensors, $\boldsymbol{\mathcal{Y}}$ (in blue). 143

10.3 Performance comparison of the proposed TSA, DPA, and ADA models. The figure columns, from left to right, correspond to the five considered datasets. The figure rows, from top to bottom, correspond to: (i) embedding dimension vs number of parameters (complexity), (ii) embedding dimension vs training accuracy (expressive power), and (iii) embedding dimension vs testing accuracy (performance). 145

11.1 The forward pass of a GTN for an order-$(3 + 2)$ input data tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times J_1 \times J_2}$, according to (11.1). 151

11.2 DNN as a GTN with $N = 0$ domain modes and $M = 1$ feature mode, according to (11.3). 152

11.3 GCN as a GTN with $N = 1$ domain mode (representing graph vertices) and $M = 1$ feature mode, according to (11.5). 153

11.4 CNN as a GTN with $N = 1$ domain mode and $M = 0$ feature mode, according to (11.7). 154

11.5 ANN as a GTN with $N = 1$ domain mode (representing time) and $M = 1$ feature mode, according to (11.9). 155

11.6 RNN as a GTN with $N = 1$ domain mode (representing time) and $M = 1$ feature mode, according to (11.14). 157

11.7 GTN model for traffic modelling. The input data tensor, $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$, contains speed measurements across $I_1 = 12$ time-steps and $I_2 = 325$ sensors. This is then contracted according to the GTN forward pass in (11.1) with a time-domain GSO, $\boldsymbol{S}^{(1)} \in \mathbb{R}^{I_1 \times I_1}$, and a sensors-domain GSO, $\boldsymbol{S}^{(2)} \in \mathbb{R}^{I_2 \times I_2}$, followed by a $tanh$ activation, $\sigma(\cdot)$. The considered GTN forward pass is then repeated 4 times. Finally, the output layer applies a time-mode contraction with a weight vector, $\boldsymbol{w} \in \mathbb{R}^{I_1}$, and a sensors-mode contraction with the sensors-mode GSO, $\boldsymbol{S}^{(2)} \in \mathbb{R}^{I_2 \times I_2}$, to yield the final output vector, $\boldsymbol{y} \in \mathbb{R}^{I_2}$. The resulting output contains the forecast of the traffic speeds for the next hour across $I_2 = 325$ sensors. 159

List of Tables

4.1 Multi-way financial data selected from Bloomberg 65
4.2 Performance comparison of the considered models. 67

5.1 Stations with largest betweenness centrality. 72
5.2 Daily average passenger flows during the morning rush hour. 76
5.3 Stations with largest net passenger out-flow and in-flow. 77
5.4 Estimation MSE of diffusion implied population using synthetic data. 79
5.5 Semi-supervised learning results from the proposed experiment. The MSE was calculated on the training and test set data using a $70 - 30\%$ split. 81

6.1 Test accuracy and Sharpe ratio, evaluated as Mean (standard deviation), of the GNN models and the base S&P500 market. 88

7.1 Experiment results for the considered models. The first row shows the MSE bewteen the ground-truth weight tensor and the estimated weight tensor. The second row shows the number of parameters as a measure of space complexity. The proposed GRTR model achieved the best results by a large margin while requiring only 200 parameters, while the other considered models over-fit the training set owing to the noisy and large-dimensional nature of the problem. 103
7.2 Experimental results for the considered models. The first row shows the test set MSE achieved by the considered models. The second row shows the required number of parameters (i.e., space complexity). The proposed GRTR model achieved the best results while requiring only 52 parameters, which is comparable to the TR and the L2TR models. Linear models once again over-fitted the training set due to the large-dimensional nature of the problem. 105

8.1 Performance comparison on the PROTEIN dataset. 121

9.1 Experiment Data Modalities 127
9.2 Performance of the Considered Models. 128
9.3 Complexity of the Considered Models 128
9.4 Sample LOB state structure with 40 dimensions for a given timestamp of AAPL stock on June 26, 2012. Source: `lobsterdata.com` 129
9.5 Sample of LOB messages for AAPL 131

9.6 Order inputs for each stock considered. 131
9.7 Model summaries for 40-dimensional LOB states times 20-timestep lookback, with number of trainable parameters and scalar output dimension for VWAP. 132
9.8 Summary of experimental results across models and stocks: MSE within a standard deviation, training time, and number of trainable parameters. 135

10.1 Expressive power measured in training accuracy (top table), and performance measured in testing accuracy (bottom table) of the considered models at a low complexity level (measured in number of parameters of around 300) across five datasets (RN, NG, DR, RA, SA). 146

11.1 Traffic prediction performance achieved by considered models. . . . 159

List of Proposed Algorithms

1 TT-SVD 42
2 CPD via ALS 46
3 TTCP 54
4 GRTR via ALS 98
5 Graph Convolution Tensor 117
6 TSA Algorithm 143

Nomenclature

Symbols

$x \in \mathbb{R}$ Scalar

$\boldsymbol{x} \in \mathbb{R}^{I_1}$ Vector of size I_1

$\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$ Matrix of size $I_1 \times I_2$

$\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ Order-N tensor of size $I_1 \times \cdots \times I_N$

$\boldsymbol{\mathcal{X}}_{i_1,\cdots,i_N} \in \mathbb{R}$ The $(i_1, \ldots, i_N)$-th entry of the order-N tensor

$\boldsymbol{X}_{\{n\}} \in \mathbb{R}^{I_n \times (I_1 \cdots I_{n-1} I_{n+1} \cdots I_N)}$ Mode-n matricization of the order-N tensor

$\bar{\boldsymbol{x}} \in \mathbb{R}^{I_1 \cdots I_N}$ Vectorization of the order-N tensor

Operators

$\star$ Convolution

$\%$ Modulo

$|| \cdot ||_F$ Frobenius norm

$\mathrm{tr}(\cdot)$ Trace

$\mathrm{vec}(\cdot)$ Vectorization

$\mathrm{mat}(\cdot, n)$ Mode-n matricization

$\mathrm{degree}(\cdot)$ Degree matrix

$(\cdot)^T$ Vector or matrix transpose

$\times_n^m$ Tensor contraction over (n, m) modes

$\circ$ Outer product

$\circledast$ Hadamard product

$\otimes$ Kronecker product

$\odot$ Khatri-Rao product

$\langle \cdot, \cdot \rangle$ Inner product

$[\![\cdot; \cdots]\!]$ Tucker product

Acronyms

ADA ADditive Attention
ALS Alternating Least Squares
ANN Attention Neural Network
ARIMA Auto-Regressive Integrated Moving Average
BCE Binary Cross-Entropy
CNN Convolutional Neural Network
CPD Canonical Polyadic Decomposition
DCG Directed Circulant Graph
DCRNN Diffusion Convolutional Recurrent Neural Network
DNN Dense Neural Network
DPA Dot Product Attention
DR Drug Review
DTW Dynamic Time Wrapping
FC-LSTM Fully-Connected Long Short-Term Memory
GCN Graph Convolutional Network
GCTN Graph Convolution Tensor Network
GNN Graph Neural Network
GRTR Graph-Regularized Tensor Regression
GRU Gated Recurrent Unit
GSO Graph Shift Operator
GSP Graph Signal Processing
GTN Graph Tensor Network
HA Historical Average
HOTTBOX Higher Order Tensor Tool BOX
HT Hierarchical Tucker
LOB Limit Order Book
LR Linear Regression
LSTM Long Short-Term Memory
MAE Mean Absolute Error
MAPE Mean Absolute Percentage Error
MERA Multiscale Entanglement Renormalization Ansatz
ML Machine Learning
MPO Matrix Product Operator
MPS Matrix Product State

MSE Mean Squared Error
NG News Groups
NLP Natural Language Processing
QTTD Quantized Tensor-Train Decomposition
RA Research Articles
RGB Red Green Blue
RGTN Recurrent Graph Tensor Network
RMSE Root Mean Square Error
RN Reuters Newswire
RNN Recurrent Neural Network
SA Spooky Authors
SRC Select-Reduce-Connect
STM Support Tensor Machine
SVD Singular Value Decomposition
SVR Support Vector Regression
TCP Tensor Contraction Product
TD Tensor Decomposition
TKD TucKer Decomposition
TN Tensor Network
TR Tensor Regression
TSA Tensorized Spectral Attention
TT Tensor-Train
TTCP Tensor-Train Contraction Product
TTD Tensor-Train Decomposition
TTNN Tensor-Train Neural Network
TTRNN Tensor-Train Recurrent Neural Network
VAR Vector Auto-Regression
VWAP Volume Weighted Average Price

Statement of Originality

I declare that the work presented in this thesis is the outcome of my research under the supervision of Prof. Danilo P. Mandic. Ideas and results from the work of others are appropriately acknowledged with the standard referencing practices employed in this discipline. The material in this thesis has not been submitted for any degree at any other academic or professional institution.

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC). Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose. When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes. Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Acknowledgments

Firstly, I would like to extend my gratitude to my supervisor, Prof. Danilo P. Mandic, for giving me the opportunity to undertake this PhD and introducing me to the fascinating worlds of tensors and graphs. As a mentor, he provided me with support and guided me through this challenging and rewarding journey, which has enabled me to grow both academically and intellectually over the last four years. Next, I would like to thank my former colleagues, Giuseppe Calvi, Bruno Scalzo, Ilya Kisil, and Shengxi Li, for their support during the early stages of my studies that laid the foundations for my doctoral work. I would also like to thank my colleagues and collaborators, Kriton Konstantinidis, Alexandros Haliassos, Alexander Jenkins, and Yuyang Miao, for many productive discussions and technical debates that led to many interesting ideas that are presented in this thesis. I am also grateful to my friends, Mattia Villani, Daniele Olmeda, Federico Sanna, and Enrico Fontana, who have kept me sane and motivated me through the ups and downs of life over the past four years. Above all, I am most grateful to my parents and my brother, who have provided me with unconditional love and relentless support throughout my life. No tensor is large enough to contain my gratitude for my family and friends.

Yao Lei Xu
2023

Publications

Journals

Peer-reviewed and published

[1] Y. L. Xu, K. Konstantinidis, and D. P. Mandic, “Graph-regularized tensor regression: A domain-aware framework for interpretable modelling of multi-way data on graphs.” *Neural Computation, in print*, 2023.

[2] I. Kisil, G. G. Calvi, K. Konstantinidis, Y. L. Xu, and D. P. Mandic, “Accelerating tensor contraction products via tensor-train decomposition,” *IEEE Signal Processing Magazine*, vol. 39, no. 5, pp. 63–70, 2022.

Submitted

[Submitted] Y. L. Xu, K. Konstantinidis, and D. P. Mandic, “Graph tensor networks: An intuitive framework for designing large-scale neural learning systems on multiple domains." *Neural Computation, in review*, 2023.

Conference Proceedings

Peer-reviewed and published

[3] T. Konstantinidis, Y. L. Xu, T. G. Constantinides, and D. P. Mandic, “A comparative study on ML-based approaches for main entity detection in financial reports,” in *Proceedings of the 24th International Conference on Digital Signal Processing (DSP)*, 2023.

[4] T. Konstantinidis, M. Golan, Y. L. Xu, T. G. Constantinides, and D. P. Mandic, “Text mining for sentiment analysis in bond portfolio construction,” in *Proceedings of the 24th International Conference on Digital Signal Processing (DSP)*, 2023.

[5] T. Konstantinidis, A. Bonet-Farres, Y. L. Xu, T. G. Constantinides, and D. P. Mandic, “Financial news classification model for NLP-based bond portfolio construction,” in *Proceedings of the 24th International Conference on Digital Signal Processing (DSP)*, 2023.

[6] A. N. Arya, Y. L. Xu, L. Stankovic, and D. P. Mandic, “Hierarchical graph learning for stock market prediction via a domain-aware graph pooling operator,” in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

[7] A. Rebello, K. Konstantinidis, Y. L. Xu, and D. P. Mandic, “Tensor completion for efficient and accurate hyperparameter optimisation in large-scale statistical learning,” in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

[8] J. Roa-Vicens, Y. L. Xu, R. Silva, and D. P. Mandic, “Graph and tensor-train recurrent neural networks for high-dimensional models of limit order books,” in *Proceedings of Third ACM International Conference on AI in Finance (ICAIF)*, 2022, pp. 207–213.

[9] K. Konstantinidis, Y. L. Xu, Q. Zhao, and D. P. Mandic, “Variational Bayesian tensor networks with structured posteriors,” in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 3638–3642.

[10] Y. L. Xu, K. Konstantinidis, S. Li, L. Stankovic, and D. P. Mandic, “Low-complexity attention modelling via graph tensor networks,” in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 3928–3932.

[11] Y. L. Xu and D. P. Mandic, “Recurrent graph tensor networks: A low-complexity framework for modelling high-dimensional multi-way sequences,” in *Proceedings of the 29th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 1795–1799.

[12] B. Scalzo, Y. L. Xu, A. G. Constantinides, and D. P. Mandic, “Graph theory for metro traffic modelling,” in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–5.

[13] Y. L. Xu, G. G. Calvi, and D. P. Mandic, “Tensor-train recurrent neural networks for interpretable multi-way financial forecasting,” in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–5.

In Preparation

[Submitted] Y. L. Xu, Y. Miao, and D. P. Mandic. “Graph convolution via convolution tensors," In *Proceedings of the 37th Conference on Neural Information Processing Systems (NIPS)*, 2023.

Workshops

Peer-reviewed and published

[14] Y. L. Xu, K. Konstantinidis, and D. P. Mandic, “A tensorized spectral attention mechanism for efficient natural language processing,” in *the Second Workshop on Quantum Tensor Networks in Machine Learning, 35th Conference on Neural Information Processing Systems (NIPS)*, 2021.

[15] K. Konstantinidis, Y. L. Xu, and D. P. Mandic, "Bayesian tensor networks with structured posteriors," in *the Second Workshop on Quantum Tensor Networks in Machine Learning, 35th Conference on Neural Information Processing Systems (NIPS)*, 2021.

[16] Y. L. Xu, K. Konstantinidis, and D. P. Mandic, "Multi-graph tensor networks," in *the First Workshop on Quantum Tensor Networks in Machine Learning, 34th Conference on Neural Information Processing Systems (NIPS)*, 2020.

Chapter 1

Introduction

1.1 Challenges of the Modern Big Data

The modern big data is naturally large- and multi-dimensional. Indeed, a typical data setting involves high-resolution data collected over time and across many sensors, leading to an explosion in dimensionality. In the context of data analytics, this results in an excessively large number of model parameters, which renders the training prone to over-fitting and impedes efficient learning - a phenomenon known as the *Curse of Dimensionality* [17]. As the world enters the era of big data, the significant expansion in both volume and range of applications, along with the concomitant increase in the variety of data sources, are posing significant challenges to traditional Machine Learning (ML) algorithms. As illustrated in Figure 1.1, these ever-expanding trends in volume, variety, veracity, and velocity constitute the 4 Vs of the modern big data [17], which have highlighted the necessity for more versatile analysis tools that offer greater opportunities for algorithmic developments and computationally faster operations than the standard flat-view matrix approaches.

Tensors have recently received tremendous attention in the data analytics community, due to their ability to mitigate the curse of dimensionality associated with the modern big data [18, 19]. Tensors are a multi-linear generalization of vectors (order-1 tensors) and matrices (order-2 tensors) to multi-way arrays (order-N tensors) [20, 21]. In particular, Tensor Decomposition (TD) techniques, similarly to their matrix decomposition counterparts, have the ability to extract latent factors in tensors, which can be employed to obtain low-rank representations of multi-way data [21, 22]. The extracted latent tensor factors can also be used to logarithmically compress large multi-dimensional tensors into an efficient low-rank format by exploiting intra- and inter-dependencies across data dimensions (tensor modes) [18, 19, 20]. This both preserves the underlying data structure and promotes interpretability, while bypassing the curse of dimensionality, as the associated computational costs reduce from an original exponential one to a linear one in the data dimensions. These desirable properties, combined with the fact that tensors can be readily constructed from vectors and matrices via tensorization [18, 19], make them particularly attractive to the data analytics communities. For instance, several TD techniques, such as Canonical Polyadic Decomposition (CPD) [23, 24], Tucker Decomposition (TKD) [25, 26], and

Tensor-Train Decomposition (TTD) [27], have successfully been applied in the context of compression [28, 29] and interpretability of neural networks [30], multi-modal learning [20], and completion of Knowledge Graphs [31, 32], just to name a few.

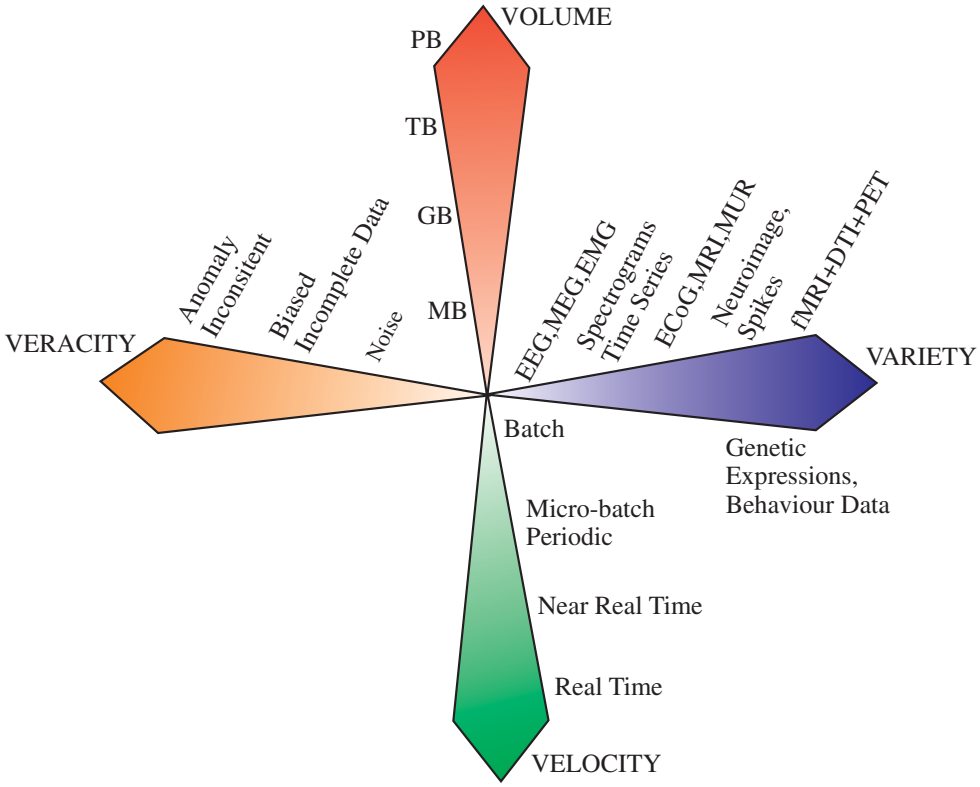


Figure 1.1: The 4 Vs of the modern big data [17]: (i) Volume - scale of data; (ii) Variety - different forms of data; (iii) Veracity - uncertainty of data; (iv) Velocity - analysis of streaming data.

The modern big data is not only characterized by exceedingly large data dimensions, but also by irregular domains with relational structures. For instance, social media platforms generate massive amounts of data on social networks, where the relationships between users and their interactions form a large graph. Similarly, transportation networks carry information on the movement of vehicles, passengers, and goods, across different locations and modes of transportation. The extraction of knowledge from these data sources must account for their underlying relational structure, which is naturally represented as a mathematical graph. However, this presents serious challenges for traditional learning algorithms, which are often by construction designed to process data on grid-like data domains that are regular and uniform in nature. For instance, classical signals, such as speech and image, reside on a regular domain where the data points are sampled at regular intervals with constant distance between samples, akin to a uniform grid [33].

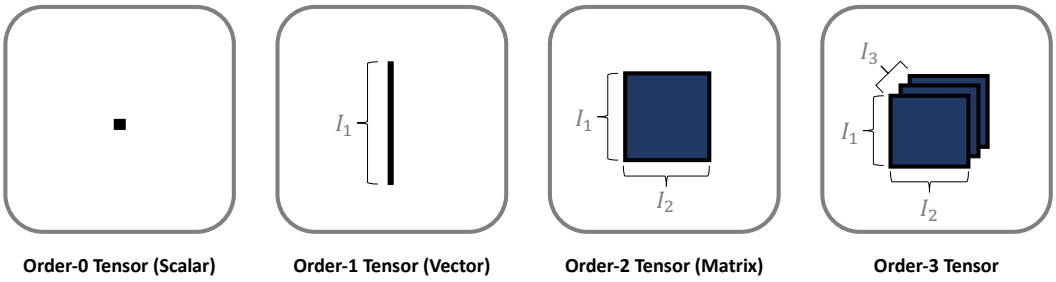


Figure 1.2: Tensors as a generalization of scalars, vectors, and matrices.

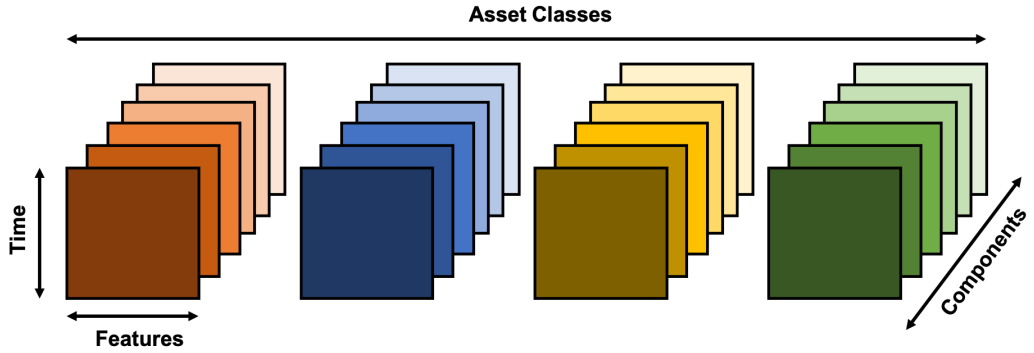


Figure 1.3: Order-4 tensor representation of the financial big data. Market data features (e.g., price) can be collected over time (e.g., daily) for each component (e.g., the Microsoft stock) of every asset class (e.g., equities).

To this end, there has been an increasing amount of research aiming at generalizing standard ML approaches to handle graph data, which arise naturally in many modern applications, as illustrated in Figure 1.4 [34]. In particular, the field of Graph Signal Processing (GSP) generalizes traditional signal processing concepts to irregular domains [35, 34, 36], achieving state-of-the-art performance in tasks where the underlying data relations can be described as a graph. Developments in GSP have led to a range of spatial and spectral based techniques that generalize the notion of frequency and locality to irregular data, allowing for the processing of signals while taking into account the underlying relational structure [33]. Concepts developed in GSP have successfully found applications in deep learning, where graph filters can be implemented across multiple Graph Neural Network (GNN) layers to incorporate graph domain information [37]. For instance, GNNs have succesfully found applications in the modelling of social networks [38], traffic prediction [39, 12], and gene inference [40].

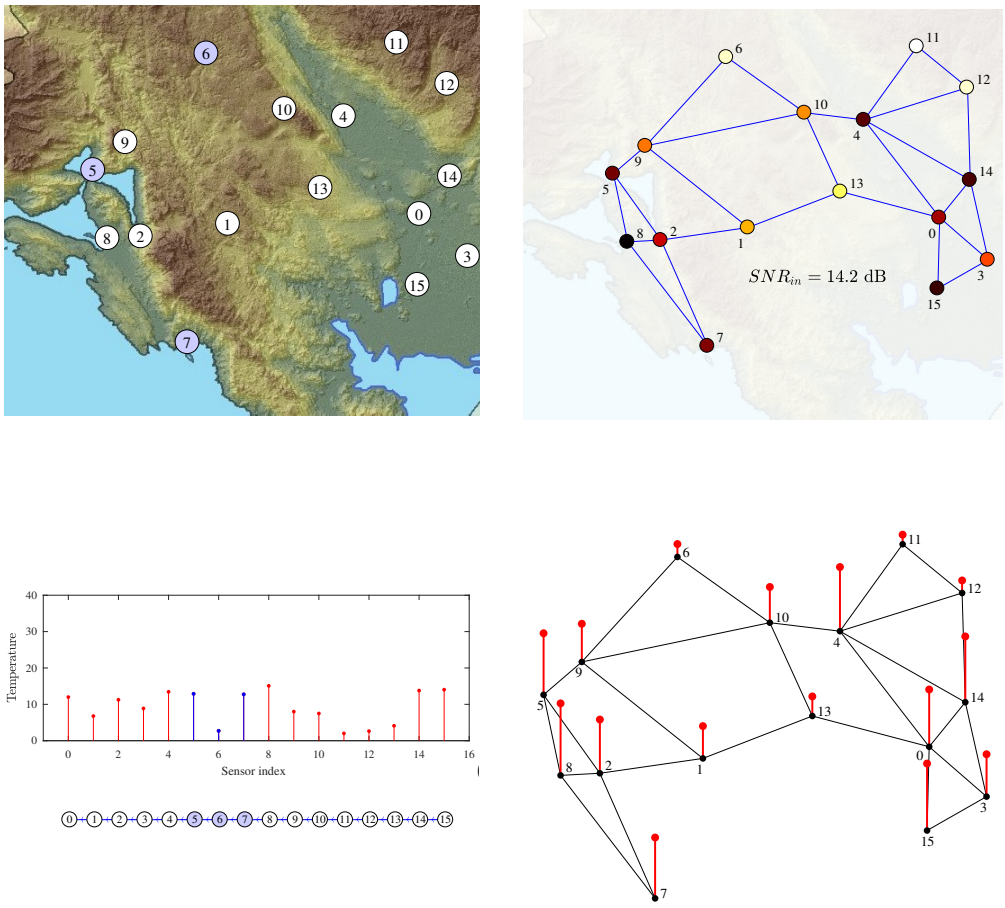


Figure 1.4: Converting temperature measurements from multiple sensors into a graph signal (adapted from [34]). The top-left panel illustrates the location of temperature measurements. The bottom-left panel illustrates the classical vector representation of the temperature signal, which is unaware of the underlying spatial structure. The top-right panel represents the top-left panel in a graph form, where the vertices represent the sensor locations while the edges denote their physical proximity. The colour of the vertices is proportional to the temperature at the associated location. The bottom-right panel illustrates the graph signal representation of temperature measurements, which is aware of the underlying geographical structure.

1.2 A Brief History of Tensors and Graphs

Tensor calculus was introduced by Italian mathematicians Gregorio Ricci-Curbastro and Tullio Levi-Civita in the early 20th century [41]. Tensors first came to prominence in physics when Albert Einstein developed the theory of general relativity to describe the properties of space-time [42]. Tensors gained traction in fields outside of physics in 1970 when Richard A. Harshman introduced the CPD [43] based on the earlier work of Frank L. Hitchcock in 1928 [44], which is also known as the CANDECOMP or PARAFAC procedure. The CPD represented large tensors as a product of smaller factor matrices, each capturing the structure within a specific data dimension. This allowed for a simple and elegant way of analyzing and interpreting large multi-dimensional data at low computational costs. This powerful method found early applications in fields such as psychometrics [45] and chemometrics [46] in the 1980s. Following the success of CPD, several other TD techniques and Tensor Networks (TNs) were developed throughout the years, such as Tucker [47], Hierarchical Tucker (HT) [48], Matrix Product States (MPS) or Tensor-Train (TT) [49], and Multiscale Entanglement Renormalization Ansatz (MERA) [50], just to name a few. These TDs have found succes in many applications, ranging from theoretical physics [51] to quantum computing [52] and deep learning [28].

The advancements in TD and TN have been instrumental in addressing challenges in data analysis, compression, signal processing, ML, and interpretability. Today, TD continues to be a vibrant area of research, with ongoing efforts to develop more efficient algorithms, extend the applicability to diverse domains, and explore new theoretical foundations for understanding the structures within large- and multi-dimensional data.

On the other hand, the study of graphs dates back to the 18th century when the mathematician Leonhard Euler addressed the famous Seven Bridges of Königsberg problem in 1736, which involved finding a path that crossed each of the city's bridges exactly once [53]. For much of its history, most of the development in graph theory was concerned with the study of graph topology and structure, led by prominent mathematicians such as Cayley [54]. Fast forward to the early 2000s, GSP came to prominence with the seminar works of David K. Hammond in 2011 [55] and David I. Shuman in 2013 [56], as increasing number of real world data applications involved processing data from sources that could naturally be represented as graphs, such as social networks, biological networks, and sensor networks. Traditional signal processing concepts, such as the Fourier transform, were generalized to process graph signals based on the earlier work of spectral graph theory that dates back to 1980 [57]. This enabled the analysis and decomposition of graph signals into frequency components that are specific to a particular graph topology, and consequently allowed for the formulation of graph filters, modulation, and convolution. By building on the mathematical foundations of GSP, graph theory has found increasingly more application in deep learning, leading to the development of GNNs such as Graph Convolutional Networks (GCNs) [58], which extended Convolutional Neural Networks (CNNs) to handle graph data.

At present, the fusion of graphs and ML continues to drive advances in diverse domains such as social network analysis, recommender systems, bioinformatics, and transportation networks, enabling researchers to extract insights from complex and irregularly connected data.

1.3 Motivation and Aims of the Thesis

Despite the promising results achieved in these fields, there is a void in literature regarding the combination of both tensors and graphs to solve ML problems. This is increasingly urgent, as the modern big data is both large- and multi-dimensional by virtue of its tensor structure, while simultaneously residing on irregular domains by virtue of its graph structure. For instance, analysis of social networks requires the processing of natural language data across multiple modes of connectivity, as illustrated in Figure 1.5 [36]. The analysis of foreign exchange data [16] requires the processing of currency information across time, countries, and market features (order-3 financial tensor), while accounting for the underlying cross-country interest rate differences (financial graph). Similarly, analysis of surveillance camera data for traffic monitoring [59] requires the processing of video data (order-4 video tensor), while accounting for the underlying road connectivity (traffic graph).

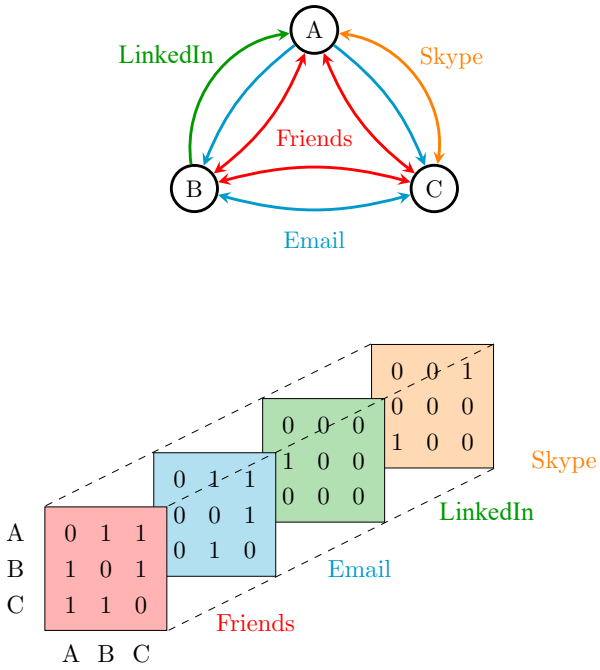


Figure 1.5: Multi-graph representation of a social network across 4 modes of communication (adapted from [36]). This can be modelled via a tensor, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{3\times3\times4}$, where (i_1, i_2, i_3)-th entry denote whether the i_1-th person is connected to the i_2-th person via the i_3-th mode of communication.

Motivated by these successful but independently developed fields, this thesis introduces several frameworks that aim at exploiting the advantages of the confluence of graphs and tensors in the context of ML. Through their joint consideration, the frameworks proposed in this thesis are shown to be able to simultaneously overcome the curse of dimensionality by virtue of tensors, while benefiting from the domain aware processing of graphs. In addition to improved performance and reduced computationnal costs, the considered frameworks are often more interpretable and flexible compared to traditional methods. It is my hope that the investigated approaches will help building a foundation for versatile data analytics tools that can tackle the challenges posed by the era of big data.

1.4 Thesis Organization

The main body of the thesis begins with **Chapter 2**, which establishes the theoretical foundation by introducing the necessary mathematical background on tensors and graphs. The remaining chapters are organized into three main parts:

- **Part I: Tensor Learning methods** explores the use of tensor techniques for data analytics. It begins with **Chapter 3**, which discusses the use of TTD for accelerating Tensor Contraction Products (TCP): A key operation underpinning multi-linear manipulation of tensors and is at the heart of all ML algorithms discussed in this thesis. Next, the TTD results from the previous chapter are extended to address the high computational costs of Recurrent Neural Networks (RNNs) in **Chapter 4**, resulting in a highly efficient Tensor-Train Recurrent Neural Network (TTRNN) model for financial modelling.

- **Part II: Graph Learning methods** begins with **Chapter 5**, which explores graph-theoretic tools in the context of metro transportation networks modelling through several intuitive examples. Next, following the promising results of tensor neural networks for financial modelling in Chapter 4, **Chapter 6** investigates the utility of GNNs in the context of short-term stock price forecasting by introducing a novel stock-pooling operator.

- **Part III: Tensor-Variate ML on Graphs** begins by examining the application of graph-theoretic concepts to enhance tensor models in **Chapter 7**. This is achieved by introducing a novel Graph-Regularized Tensor Regression (GRTR) framework, whereby domain knowledge about intra-modal relations is incorporated into the model in the form of a graph Laplacian matrix. This is then used as a regularization tool to promote a physically meaningful structure within Tensor Regression (TR) models. Next, instead of enhancing tensor models through graph-theoretic concepts, **Chapter 8** applies tensor-theoretic concepts to enhance graph convolutions in terms of efficiency, flexibility, and robustness. A joint treatment of tensors, graphs, and neural networks is next discussed in **Chapter 9**, where a multi-linear graph filter for approximating RNNs in TN form is introduced, resulting in a novel Recurrent Graph Tensor Network (RGTN) framework. The RGTN is then extended in **Chapter 10** to perform highly

efficient attention modelling: A key mechanism at the core of state-of-the-art Natural Language Processing (NLP) models. The results from the previous two chapters are then extended in **Chapter 11**, resulting in the generalized Graph Tensor Network (GTN) framework. The proposed framework is shown to be general enough to include many modern neural network architectures as special cases, and flexible enough to handle data on any and many data domains. This opens avenues for the design of new and efficient neural networks which are not possible to arrive at otherwise.

Finally, the main findings of the this work are summarized in **Chapter 12**. The high-level organization of the thesis is illustrated in Figure 1.6 towards the end of this chapter.

1.5 Contributions

The main contributions of each individual chapter are summarised and presented below, together with their associated publications.

Part I: Tensor Learning Methods

- **Chapter 3: Tensor Contraction via Tensor-Train Decomposition**

  A key tool underpinning multi-linear manipulation of tensors and TNs is the standard TCP, which is at the heart of all ML algorithms discussed in this thesis. However, depending on the dimensionality of the underlying tensors, the TCP also comes at the price of high computational complexity in tensor manipulation. In this chapter, we resort to diagrammatic TN manipulation to calculate such products in an efficient and computationally tractable manner, by making use of TTD. This has rendered the underlying concepts easy to perceive, thereby enhancing intuition of the associated underlying operations, while preserving mathematical rigour. In addition to bypassing the cumbersome mathematical multi-linear expressions, the proposed TTCP model is shown to accelerate significantly the underlying computational operations, as it is independent of tensor order and linear in the tensor dimension, as opposed to performing the full computations through the standard approach (exponential in tensor order).

  Relevant publications:

  [2] I. Kisil, G. G. Calvi, K. Konstantinidis, Y. L. Xu, and D. P. Mandic, "Accelerating tensor contraction products via tensor-train decomposition," *IEEE Signal Processing Magazine*, vol. 39, no. 5, pp. 63–70, 2022.

- **Chapter 4: Tensor-Train Recurrent Neural Networks for Finance**

  In this chapter, the TTD results from the previous chapter are extended to address the high complexity costs of RNNs. RNNs represent the de facto standard ML tool for sequence modelling, owing to their expressive power and memory. However, when dealing with large dimensional data, the corresponding

exponential increase in the number of parameters imposes a computational bottleneck. The necessity to equip RNNs with the ability to deal with the curse of dimensionality, such as through the parameter compression ability inherent to tensors, has led to the development of the TTRNN. Despite achieving promising results in many applications, the full potential of the TTRNN is yet to be explored in the context of interpretable financial modelling, a notoriously challenging task characterized by multi-modal data with low signal-to-noise ratio. To address this issue, we investigate the potential of TTRNN in the task of financial forecasting of currencies. We show, through the analysis of TT-factors, that the physical meaning underlying TD, enables the TTRNN model to aid the interpretability of results, thus mitigating the notorious "black-box" issue associated with neural networks. Furthermore, simulation results highlight the regularization power of TT decomposition, demonstrating the superior performance of TTRNN over its uncompressed RNN counterpart and other tensor forecasting methods.

Relevant publications:

[13] Y. L. Xu, G. G. Calvi, and D. P. Mandic, "Tensor-train recurrent neural networks for interpretable multi-way financial forecasting," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–5.

Part II: Graph Learning Methods

- **Chapter 5: Graph Theory for Metro Traffic Modelling**

  Part II of this thesis commences by introducing various graph theory concepts, accompanied by real-life applications that offer intuitive insights. In particular, a unifying graph theoretic framework for the modelling of metro transportation networks is proposed. This is achieved by first introducing a basic graph framework for the modelling of the London underground system from a diffusion law point of view. This forms a basis for the analysis of both station importance and their vulnerability, whereby the concept of graph vertex centrality plays a key role. We next explore k-edge augmentation of a graph topology, and illustrate its usefulness both for improving the network robustness and as a planning tool. Upon establishing the graph theoretic attributes of the underlying graph topology, we proceed to introduce models for processing data on such a metro graph. Commuter movement is shown to obey the Fick's law of diffusion, where the graph Laplacian provides an analytical model for the diffusion process of commuter population dynamics. Finally, we also explore the application of modern deep learning models, such as GNNs and hyper-graph GNNs, as general purpose models for the modelling and forecasting of underground data, especially in the context of the morning and evening rush hours. Comprehensive simulations including the passenger in- and out-flows during the morning rush hour in London demonstrates the advantages of the graph models in metro planning and traffic management, a formal mathematical approach with wide economic implications.

Relevant publications:

[12] B. Scalzo, Y. L. Xu, A. G. Constantinides, and D. P. Mandic, "Graph theory for metro traffic modelling," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–5.

- **Chapter 6: Hierarchical Graph Pooling for Finance**

  As previously discussed in Chapter 4, the use of tensor techniques can substantially enhance neural network performance for financial modelling. Similarly, this chapter investigates the utility of GNNs in the context of short-term stock price forecasting. In particular, a finance-specific graph pooling operation, referred to as *StockPool*, is introduced to efficiently coarsen the stock graph. This is achieved by employing domain knowledge to cluster stocks, depending on some task-specific characteristics (e.g. industries and sub-industries). Unlike fully end-to-end learnable graph pooling strategies (e.g. differentiable pooling and MinCUT pooling), such a deterministic pooling operator is considerably more computationally efficient and thus scalable to larger stock graphs. Experimentations on the S&P500 stock index demonstrate that the StockPool operator outperforms existing graph pooling strategies on the prediction of price movements. Finally, different graph pooling methods are utilized to create a set of highly uncorrelated GNN models; these are used to construct a graph ensemble model with an improved performance.

  Relevant publications:

  [6] A. N. Arya, Y. L. Xu, L. Stankovic, and D. P. Mandic, "Hierarchical graph learning for stock market prediction via a domain-aware graph pooling operator," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

Part III: Tensor-Variate Machine Learning on Graphs

- **Chapter 7: Graph-Regularized Tensor Regression**

  Part III of this thesis begins by examining the application of graph-theoretic concepts to enhance tensor models. This is increasingly essential due to the exceedingly large and multi-dimensional data sources that define modern big data. This represents a challenge for traditional ML models, as the number of model parameters needed to process such data grows exponentially with the data dimensions; an effect known as the curse of dimensionality. Recently, TD techniques have shown promising results in reducing the computational costs associated with large-dimensional models while achieving comparable performance. However, such tensor models are often unable to incorporate the underlying domain knowledge when compressing high-dimensional models. To this end, we introduce a novel GRTR framework, whereby domain knowledge about intra-modal relations is incorporated into the model in the form of a graph Laplacian matrix. This is then used as a regularization tool to promote a physically meaningful structure within the model parameters. By virtue of

tensor algebra, the proposed framework is shown to be fully interpretable, both coefficient-wise and dimension-wise. The GRTR model is validated in a multi-way regression setting and compared against competing models, and is shown to achieve improved performance at reduced computational costs. Detailed visualizations are provided to help the reader gain an intuitive understanding of the employed tensor operations.

Relevant publications:

[1] Y. L. Xu, K. Konstantinidis, and D. P. Mandic, "Graph-regularized tensor regression: A domain-aware framework for interpretable modelling of multi-way data on graphs." *Neural Computation, in print*, 2023.

- **Chapter 8: Graph Convolution via Convolution Tensors**

  The performance of tensor models can be greatly enhanced through graph-theoretic concepts, as discussed in Chapter 7. Similarly, this chapter discusses the use of tensor-theoretic concepts to improve graph convolutions. Convolution is the linchpin operation of modern deep learning. However, traditional convolution does not generalize to graphs due to the irregular structure of node neighbourhoods. To this end, we introduce the Graph Convolution Tensor Network (GCTN), which generalizes the notion of convolution tensor to general graphs. The proposed framework offers several advantages compared to existing methods in terms of efficiency, scalability, robustness, flexibility, and performance. In addition, the GCTN is general enough to include the classical CNN as a special case of convolution over circulant graphs. The benefits of the proposed framework are demonstrated through experiments, achieving state-of-the-art performance on the protein classification dataset.

  Relevant publications:

  [Submitted] Y. L. Xu, Y. Miao, and D. P. Mandic. "Graph convolution via convolution tensors," In *Proceedings of the 37th Conference on Neural Information Processing Systems (NIPS)*, 2023.

- **Chapter 9: Recurrent Graph Tensor Networks**

  This chapter analyzes RNNs through a joint consideration of tensors and graphs. RNNs are among the most successful ML models for sequence modelling, but tend to suffer from an exponential increase in the number of parameters when dealing with large multidimensional data. To this end, we develop a multi-linear graph filter framework for approximating the modelling of hidden states in RNNs, which is embedded in a TN architecture to improve modelling power and reduce parameter complexity, resulting in a novel Recurrent Graph Tensor Network (RGTN). The proposed framework is validated through several multi-way sequence modelling tasks and benchmarked against traditional RNNs. By virtue of the domain aware information processing of graph filters and the expressive power of TNs, we show that the proposed RGTN is capable of not only out-performing standard RNNs, but also mitigating the curse of dimensionality associated with traditional RNNs, demonstrating superior properties in terms of performance and complexity.

Relevant publications:

[11] Y. L. Xu and D. P. Mandic, "Recurrent graph tensor networks: A low-complexity framework for modelling high-dimensional multi-way sequences," in *Proceedings of the 29th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 1795–1799.

[8] J. Roa-Vicens, Y. L. Xu, R. Silva, and D. P. Mandic, "Graph and tensor-train recurrent neural networks for high-dimensional models of limit order books," in *Proceedings of Third ACM International Conference on AI in Finance (ICAIF)*, 2022, pp. 207–213.

- **Chapter 10: Attention Graph Tensor Networks**

  Building upon the RGTNs introduced in the previous chapter, this chapter explores their application in the context of attention modelling. The attention mechanism is at the core of state-of-the-art NLP models, owing to its ability to focus on the most contextually relevant part of a sequence. However, current attention models rely on "flat-view" matrix methods to process sequence of tokens embedded in vector spaces, resulting in exceedingly high parameter complexity for practical applications. To this end, we introduce a novel *Tensorized Spectral Attention* (TSA) mechanism, which leverages on the GTN framework to efficiently process tensorized token embeddings via attention based spectral graph filters. By virtue of multi-linear algebra, such tensorized token embeddings are shown to effectively bypass the curse of dimensionality, reducing the parameter complexity of the attention mechanism from exponential to linear in the weight matrix dimensions. Furthermore, the graph formulation of the attention domain enables the processing of tensorized embeddings through spectral graph convolution filters, which further increases its expressive power. The benefits of the TSA are demonstrated through five benchmark NLP experiments, where the proposed mechanism is shown to achieve better or comparable results against traditional attention models, while incurring drastically lower parameter complexity.

  Relevant publications:

  [10] Y. L. Xu, K. Konstantinidis, S. Li, L. Stankovic, and D. P. Mandic, "Low-complexity attention modelling via graph tensor networks," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 3928–3932.

  [14] Y. L. Xu, K. Konstantinidis, and D. P. Mandic, "A tensorized spectral attention mechanism for efficient natural language processing," in *the Second Workshop on Quantum Tensor Networks in Machine Learning, 35th Conference on Neural Information Processing Systems (NIPS)*, 2021.

- **Chapter 11: Generalized Graph Tensor Networks**

  Despite the omnipresence of tensors and tensor operations in modern deep learning, the use of tensor algebra to formally design and describe neural networks is still under-explored within the deep learning community. To this

end, we introduce the GTN framework, an intuitive yet rigorous graphical framework for systematically designing and implementing large-scale neural learning systems on both regular and irregular domains. This is achieved by extending the results from the previous two chapters to a more general setting through the Tucker product. The proposed framework is shown to be general enough to include many popular architectures as special cases, and flexible enough to handle data on any and many data domains. This opens avenues for the design of new and efficient neural networks which are not possible to arrive at otherwise. The power and flexibility of the proposed framework is demonstrated through real-data experiments.

Relevant publications:

[16] Y. L. Xu, K. Konstantinidis, and D. P. Mandic, "Multi-graph tensor networks," in *the First Workshop on Quantum Tensor Networks in Machine Learning, 34th Conference on Neural Information Processing Systems (NIPS)*, 2020.

[Submitted] Y. L. Xu, K. Konstantinidis, and D. P. Mandic, "Graph tensor networks: An intuitive framework for designing large-scale neural learning systems on multiple domains." *Neural Computation, in review*, 2023.

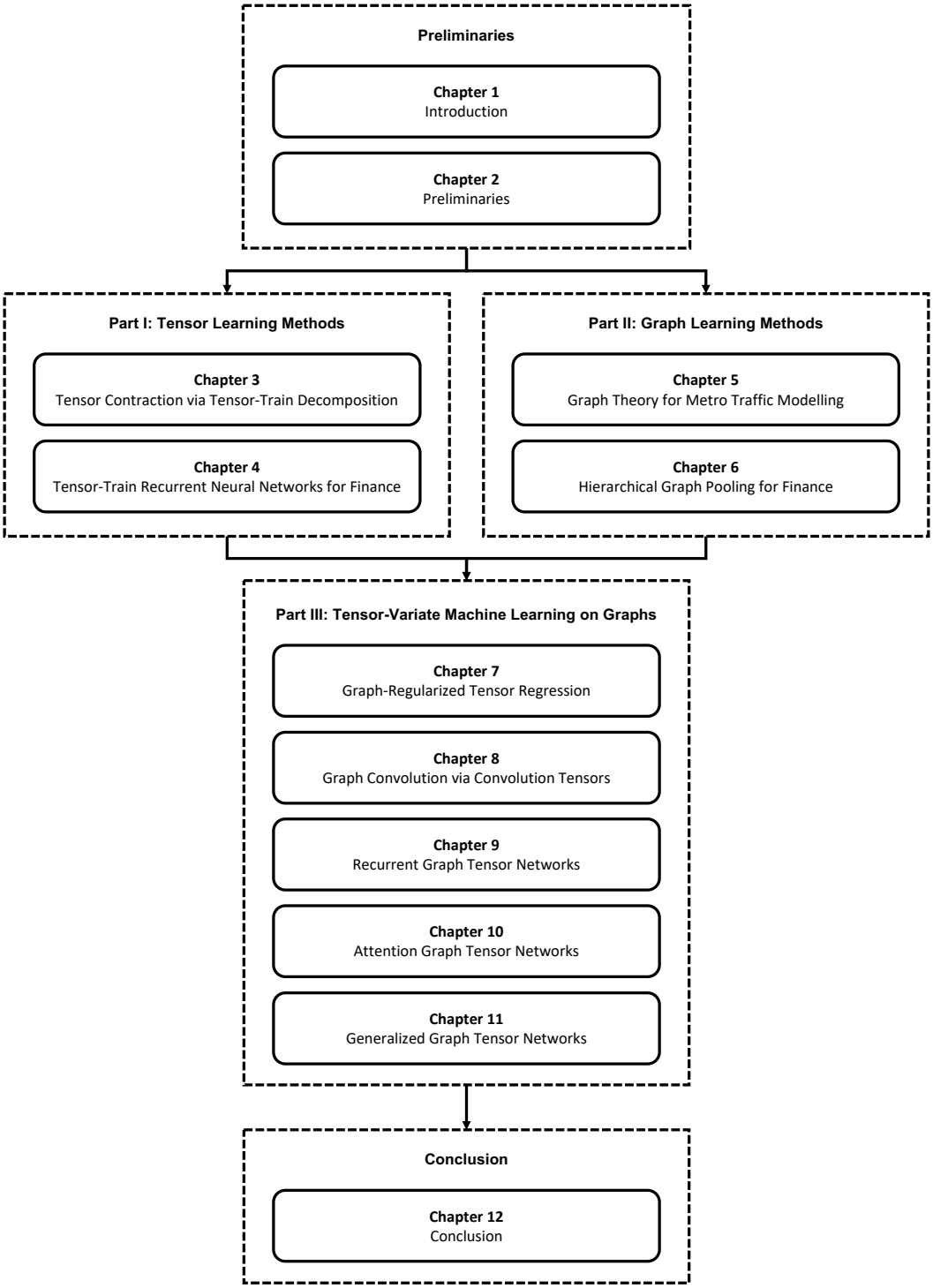


Figure 1.6: Thesis Structure.

Chapter 2

Preliminaries

This chapter establishes the theoretical foundation of the thesis. The first part presents the tensor mathematics necessary to follow this work. We refer the readers to [18, 19] for an in-depth treatment of tensor algebra, TD, and TNs. The tensor indices in this thesis are grouped according to the little-endian convention [60]. The second part of this chapter introduces the necessary GSP concepts. We refer the readers to [35, 34, 36] for an in-depth treatment of graph ML techniques.

2.1 Preliminaries on Tensors

Order-N Tensor: An order-N real-valued tensor is a multi-dimensional array, denoted by a calligraphic font, e.g., $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, where N is the order of the tensor (i.e., number of modes), and I_n ($1 \leq n \leq N$) is the size (i.e., the dimension) of its n-th mode. Matrices (denoted by bold capital letters, e.g., $\boldsymbol{A} \in \mathbb{R}^{I_1 \times I_2}$) can be seen as order-2 tensors ($N = 2$), vectors (denoted by bold lower-case letters, e.g., $\boldsymbol{a} \in \mathbb{R}^{I}$) can be seen as order-1 tensors ($N = 1$), and scalars (denoted by lower-case letters, e.g., $a \in \mathbb{R}$) are order-0 tensors ($N = 0$).

Tensor Entries: The $(i_1, \cdots, i_N)$-th entry of an order-N tensor, $\boldsymbol{\mathcal{A}}$, is denoted by $\boldsymbol{\mathcal{A}}[i_1, \ldots, i_N] = \boldsymbol{\mathcal{A}}_{i_1, \cdots, i_N} \in \mathbb{R}$, where $i_n = 1, \ldots, I_n$ for $n = 1, \ldots, N$. A tensor fiber is a vector of tensor entries obtained by fixing all but one index of the original tensor (e.g., $\boldsymbol{\mathcal{A}}[:, i_2, \ldots, i_N] = \boldsymbol{\mathcal{A}}_{:, i_2, \ldots, i_N} \in \mathbb{R}^{I_1}$). Similarly, a tensor slice is a matrix of tensor entries obtained by fixing all but two indices of the original tensor (e.g., $\boldsymbol{\mathcal{A}}[:, :, i_3, \ldots, i_N] = \boldsymbol{\mathcal{A}}_{:, :, i_3, \ldots, i_N} \in \mathbb{R}^{I_1 \times I_2}$).

Tensorization: A vector, $\boldsymbol{a} \in \mathbb{R}^{I_1 \cdots I_N}$, can be tensorized (or folded) into an order-N tensor, $\boldsymbol{\mathcal{A}}$, with the relation between their entries defined by

$$\boldsymbol{\mathcal{A}}\Big[i_1, \ldots, i_N\Big] = \boldsymbol{a}_i \tag{2.1}$$

where $i = 1 + \sum_{n=1}^{N} (i_n - 1) \prod_{n'=1}^{n-1} I_{n'}$.

Matricization (Mode-n unfolding): Mode-n matricization of a tensor, $\text{mat}\,(\boldsymbol{\mathcal{A}}, n) = \boldsymbol{A}_{\{n\}} \in \mathbb{R}^{I_n \times (I_1 \cdots I_{n-1} I_{n+1} \cdots I_N)}$, is a procedure of mapping the elements from a multidimensional array to a two-dimensional array (matrix). Conventionally, such procedure is associated with stacking mode-n fibers (modal vectors) as column vectors of the resulting matrix. For instance, the mode-1 unfolding of $\boldsymbol{\mathcal{A}}$ is represented as $\text{mat}\,(\boldsymbol{\mathcal{A}}, 1) = \boldsymbol{A}_{\{1\}} \in \mathbb{R}^{I_1 \times (I_2 \cdots I_N)}$, where the subscript, $\{1\}$, denotes the mode of matricization, and is given by

$$\boldsymbol{A}_{\{1\}}\left[i_1, \overline{i_2 \ldots i_N}\right] = \boldsymbol{\mathcal{A}}_{i_1,\ldots,i_N} \tag{2.2}$$

where the overlined subscripts refer to the little-endian indexing [60], given by:

$$\overline{i_2 \ldots i_N} = 1 + \sum_{n=2}^{N} \left[(i_n - 1) \prod_{n'=2}^{n-1} I_{n'} \right] \tag{2.3}$$

Figure 2.1 illustrates the matricization operation for an order-3 tensor.

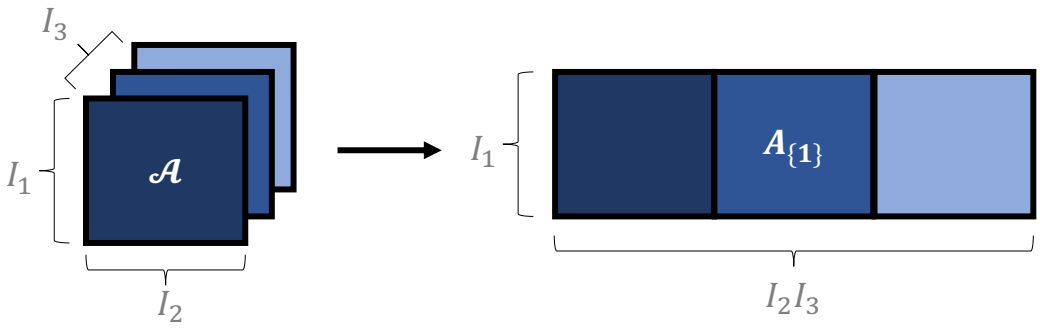


Figure 2.1: Matricization of an order-3 tensor with respect to mode-1.

Vectorization: Given an order-N tensor, $\boldsymbol{\mathcal{A}}$, its *vectorization* reshapes the multidimensional array into a large vector, $\text{vec}\,(\boldsymbol{\mathcal{A}}) = \bar{\boldsymbol{a}} \in \mathbb{R}^{I_1 \cdots I_N}$. Figure 2.2 illustrates the vectorization operation for an order-2 tensor.

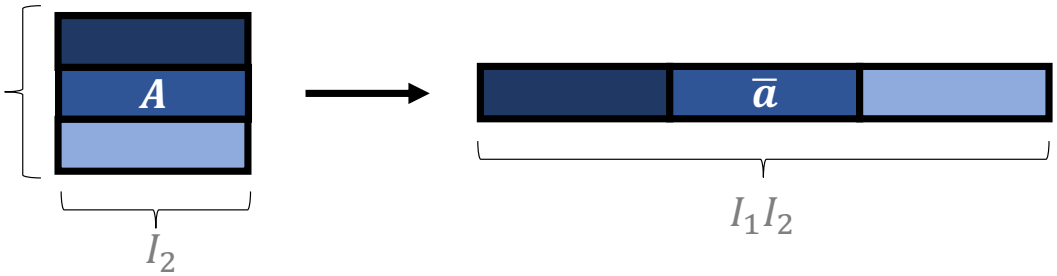


Figure 2.2: Vectorization of an order-2 tensor.

Hadamard Product: The Hadamard (element-wise) product between two order-N tensors with the same mode sizes, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, denoted by $\circledast$, yields a tensor, $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}} \circledast \boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, where $\boldsymbol{\mathcal{C}}_{i_1,\ldots,i_N} = \boldsymbol{\mathcal{A}}_{i_1,\ldots,i_N} \boldsymbol{\mathcal{B}}_{i_1,\ldots,i_N}$.

Outer product: The *outer product* of vectors $\boldsymbol{a} \in \mathbb{R}^I$ and $\boldsymbol{b} \in \mathbb{R}^J$ is given by $\boldsymbol{C} = \boldsymbol{a} \circ \boldsymbol{b} \in \mathbb{R}^{I \times J}$, with $\boldsymbol{C}_{i,j} = \boldsymbol{a}_i \boldsymbol{b}_j$.

Kronecker product: A (left) Kronecker product between two tensors, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$, denoted by $\otimes$, yields a tensor of the same order, $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{I_1 J_1 \times \cdots \times I_N J_N}$, with entries $\boldsymbol{\mathcal{C}}_{\overline{i_1 j_1},\ldots,\overline{i_N j_N}} = \boldsymbol{\mathcal{A}}_{i_1 \ldots i_N} \boldsymbol{\mathcal{B}}_{j_1 \ldots j_N}$, where $\overline{i_n j_n} = j_n + (i_n - 1) J_n$ [17]. For the special case of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, the Kronecker product yields a block-matrix:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} \boldsymbol{A}_{11}\boldsymbol{B} & \cdots & \boldsymbol{A}_{1I_2}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{I_1 1}\boldsymbol{B} & \cdots & \boldsymbol{A}_{I_1 I_2}\boldsymbol{B} \end{bmatrix} \tag{2.4}$$

Khatri-Rao product: The *Khatri-Rao product* of two matrices $\boldsymbol{A} = [\boldsymbol{a}^{(1)}, \ldots, \boldsymbol{a}^{(R)}] \in \mathbb{R}^{I \times R}$ and $\boldsymbol{B} = [\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(R)}] \in \mathbb{R}^{J \times R}$ is denoted by $\boldsymbol{C} = \boldsymbol{A} \odot \boldsymbol{B} = [\boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(R)}] \in \mathbb{R}^{IJ \times R}$, where the columns $\boldsymbol{c}^{(r)} = \boldsymbol{a}^{(r)} \otimes \boldsymbol{b}^{(r)}$, $1 \leq r \leq R$.

Tensor contraction: The *contraction* of an order-N tensor, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, and an order-$M$ tensor $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$, over the n^{th} and m^{th} modes respectively, where $I_n = J_m$, results in $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N \times J_1 \times \cdots \times J_{m-1} \times J_{m+1} \times J_M}$, with entries

$$\boldsymbol{\mathcal{C}}_{i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N,j_1,\ldots,j_{m-1},j_{m+1},\ldots,j_M} = \sum_{i_n=1}^{I_n} \boldsymbol{\mathcal{A}}_{i_1,\ldots,i_{n-1},i_n,i_{n+1},\ldots,i_N} \boldsymbol{\mathcal{B}}_{j_1,\ldots,j_{m-1},i_n,j_{m+1},\ldots,j_M} \tag{2.5}$$

A $(2,1)$-tensor contraction between two order-2 tensors, $\boldsymbol{A} \in \mathbb{R}^{I_1 \times I_2}$ and $\boldsymbol{B} \in \mathbb{R}^{J_1 \times J_2}$, where $I_2 = J_1$, is equivalent to a standard matrix multiplication, $\boldsymbol{C} = \boldsymbol{A} \times_2^1 \boldsymbol{B} = \boldsymbol{A}\boldsymbol{B} \in \mathbb{R}^{I_1 \times J_2}$. Similarly, a $(2,1)$-tensor contraction between an order-2 tensor, $\boldsymbol{A} \in \mathbb{R}^{I_1 \times I_2}$, and an order-1 tensor, $\boldsymbol{b} \in \mathbb{R}^{J_1}$, where $I_2 = J_1$, is equivalent to a standard matrix-by-vector multiplication, $\boldsymbol{c} = \boldsymbol{A} \times_2^1 \boldsymbol{b} = \boldsymbol{A}\boldsymbol{b} \in \mathbb{R}^{I_1}$.

Tensor inner product: The *inner product* of two order-N tensors, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, results in a scalar, $c = \langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle = \sum_{i_1,\ldots,i_N} \boldsymbol{\mathcal{A}}_{i_1,\ldots,i_N} \boldsymbol{\mathcal{B}}_{i_1,\ldots,i_N} \in \mathbb{R}$.

Rank-1 tensor: A order-N tensor, $\boldsymbol{\mathcal{A}}$, is said to be rank-$R = 1$, if it can be expressed as an outer product of N vectors, $\boldsymbol{\mathcal{A}} = \boldsymbol{a}^{(1)} \circ \cdots \circ \boldsymbol{a}^{(N)} = \bigcirc_{n=1}^{N} \boldsymbol{a}^{(n)}$, where $\boldsymbol{a}^{(n)} \in \mathbb{R}^{I_n}$.

Rank-1 tensor inner product: The inner product between two order-N tensors, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, can be expressed as a sequence of tensor contractions, $\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle = \boldsymbol{\mathcal{A}} \times_1^1 \boldsymbol{b}^{(1)} \times_2^1 \cdots \times_N^1 \boldsymbol{b}^{(N)}$, if the tensor $\boldsymbol{\mathcal{B}}$ is of rank-1, such that $\boldsymbol{\mathcal{B}} = \bigcirc_{n=1}^{N} \boldsymbol{b}^{(n)}$.

Mode-n Product: A mode-n product, $\mathcal{C} = \mathcal{A} \times_n^2 \boldsymbol{B}$, between an order-$N$ tensor, $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, and a matrix, $\boldsymbol{B} \in \mathbb{R}^{J \times I_n}$, yields another order-$N$ tensor, $\mathcal{C} \in \mathbb{R}^{I_1 \times I_{n-1} \times J \times I_{n+1} \times \cdots \times J_N}$. This is equivalent to the matrix multiplication, $\boldsymbol{C}_{\{n\}} = \boldsymbol{B}\boldsymbol{A}_{\{n\}}$, followed by a tensorization operation, as illustrated in Figure 2.3.

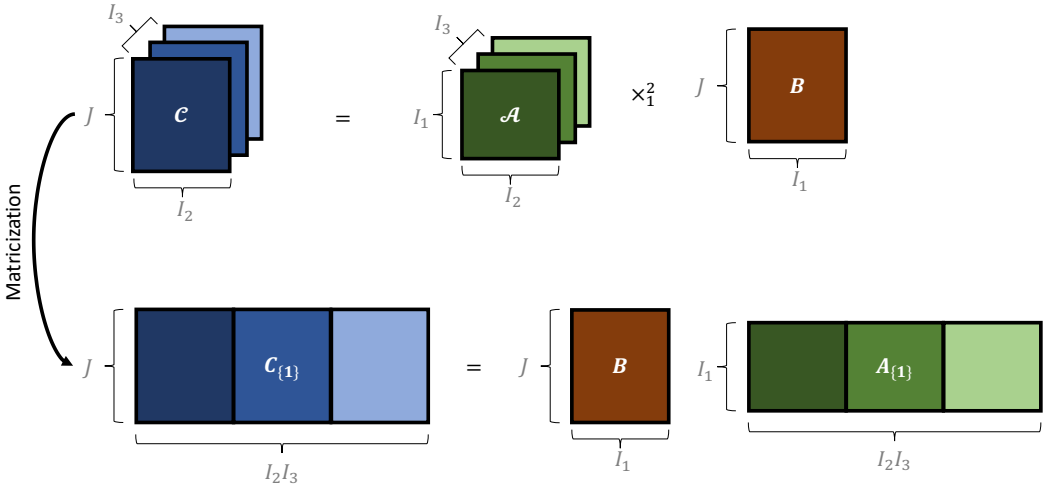


Figure 2.3: Tensor (top) and matricized (bottom) view of a mode-$n = 1$ product between an order-3 tensor, $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, and matrix, $\boldsymbol{B} \in \mathbb{R}^{J \times I_1}$, to yield, $\mathcal{C} \in \mathbb{R}^{J \times I_2 \times I_3}$.

Tensor-by-vector product: A tensor-by-vector product is a special case of tensor contraction between an order-N tensor, $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, and an order-1 tensor, $\boldsymbol{b} \in \mathbb{R}^{I_n}$, which results in an order-$(N-1)$ tensor, $\mathcal{C} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N}$, which is exactly 1 order smaller than $\mathcal{A}$. This product can be expressed as $\mathcal{C} = \mathcal{A} \times_n^1 \boldsymbol{b} = \sum_{i_n=1}^{I_n} \boldsymbol{b}_{i_n} \mathcal{A}_{:,\cdots,:,i_n,:,\cdots,:}$, where $\mathcal{A}_{:,\cdots,:,i_n,:,\cdots,:} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N}$ is an order-$(N-1)$ tensor obtained by fixing the n-th mode. This operation is mathematically equivalent to performing the following steps in a sequence: (i) Matricize the tensor, $\boldsymbol{A}_{\{n\}} = \text{mat}\,(\mathcal{A}, n)$; (ii) Perform a linear combination (matrix-by-vector multiplication) in the n-th subspace, $\boldsymbol{C}_{\{n\}} = \boldsymbol{A}_{\{n\}}^T \boldsymbol{b}$; (iii) Tensorize the result, $\mathcal{C} = \text{ten}\left(\boldsymbol{C}_{\{n\}}\right)$. Therefore, the tensor-by-vector product can be interpreted as performing a linear combination in the n-th subspace.

Example 1 *A $(3,1)$-tensor contraction between an order-3 tensor, $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, and a vector, $\boldsymbol{b} \in \mathbb{R}^{I_3}$, yields a matrix, $\boldsymbol{C} \in \mathbb{R}^{I_1 \times I_2}$, such that $\boldsymbol{C} = \mathcal{A} \times_3^1 \boldsymbol{b} = \sum_{i_3=1}^{I_3} \boldsymbol{b}_{i_3} \mathcal{A}_{:,:,i_3}$, where $\mathcal{A}_{:,:,i_3} \in \mathbb{R}^{I_1 \times I_2}$ are mode-3 tensor slices of $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$.*

Tucker Product: A Tucker product of the form, $\mathcal{C} = [\![\mathcal{A}; \boldsymbol{B}^{(1)}, \ldots, \boldsymbol{B}^{(N)}]\!]$, between an order-$N$ tensor, $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, and N matrices, $\boldsymbol{B}^{(n)} \in \mathbb{R}^{J_n \times I_n}$, $n = 1, \ldots, N$, yields another order-N tensor, $\mathcal{C} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$, through N mode-n products, $\mathcal{C} = \mathcal{A} \times_1^2 \boldsymbol{B}^{(1)} \times_2^2 \cdots \times_N^2 \boldsymbol{B}^{(N)}$. The Tucker product can also be written in vectorized form as $\text{vec}\,(\mathcal{C}) = \left(\bigotimes_{n=1}^N \boldsymbol{B}^{(n)}\right) \text{vec}\,(\mathcal{A})$, where $\bigotimes_{n=1}^N \boldsymbol{B}^{(n)} = \boldsymbol{B}^{(1)} \otimes \cdots \otimes \boldsymbol{B}^{(N)}$.

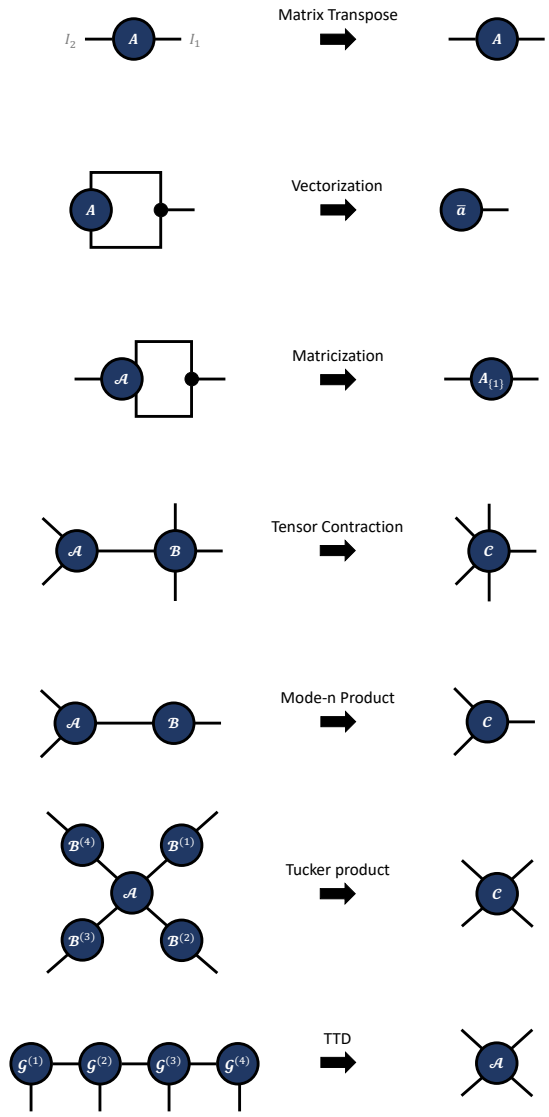


Figure 2.4: TN diagrams for basic tensor operations: (1) Matrix transpose from $\boldsymbol{A} \in \mathbb{R}^{I_1 \times I_2}$ to $\boldsymbol{A}^T \in \mathbb{R}^{I_2 \times I_1}$; (2) Matrix vectorization from $\boldsymbol{A} \in \mathbb{R}^{I_1 \times I_2}$ to $\bar{\boldsymbol{a}} \in \mathbb{R}^{I_1 I_2}$; (3) Tensor matricization from $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ to $\boldsymbol{A}_{\{1\}} \in \mathbb{R}^{I_1 \times I_2 I_3}$; (4) Tensor contraction, $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}} \times_1^4 \boldsymbol{\mathcal{B}}$, with $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{J_1 \times J_2 \times J_3 \times J_4}$ to yield $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{I_2 \times I_3 \times J_1 \times J_2 \times J_3}$; (5) Mode-$n$ product, $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}} \times_2^2 \boldsymbol{B}$, with $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and $\boldsymbol{B} \in \mathbb{R}^{I_1 \times J}$ to yield $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{I_2 \times I_3 \times J}$; (6) Tucker product, $\boldsymbol{\mathcal{C}} = [\![\boldsymbol{\mathcal{A}}; \boldsymbol{B}^{(1)}, \boldsymbol{B}^{(2)}, \boldsymbol{B}^{(3)}, \boldsymbol{B}^{(4)}]\!]$, with $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4}$ and $\boldsymbol{B}^{(n)} \in \mathbb{R}^{I_n \times J_n}$ for $n = 1, \ldots, 4$ to yield $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{J_1 \times J_2 \times J_3 \times J_4}$; (7) TTD, $\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{G}}^{(1)} \times_4^1 \boldsymbol{\mathcal{G}}^{(2)} \times_4^1 \boldsymbol{\mathcal{G}}^{(3)} \times_4^1 \boldsymbol{\mathcal{G}}^{(4)}$, with $\boldsymbol{\mathcal{G}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times J_n \times R_n}$ for $n = 1, \ldots, 4$ to yield $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4}$.

Tensor Networks: Tensor operations are often associated with an overwhelming indexing and computational requirements. These quickly become unmanageable for larger TNs, whereby multiple tensor operations are carried out across a large number of tensors. Manipulation of such expressions is prone to errors and prohibitive to manipulation of higher order tensors. To this end, the TN diagram is adopted to visualize tensor operations in an intuitive yet mathematically rigorous way. In a TN diagram, a vertex with N open edges represents an order-N tensor, while a pair of vertices connected through a common edge represents a tensor contraction between two tensors over their modes of common size. Basic tensor operations in the TN notation are illustrated in Figure 2.4.

Tensor-Train Decomposition: TTD [61] [49] was introduced by Oseledets to help mitigate the computational bottlenecks that arise from the curse of dimensionality, as tensor algorithms can become intractable for high order tensors. The most common form of TT is the MPS, introduced in the quantum physics community [17], which applies the TT Singular Value Decomposition (TT-SVD) algorithm described in Algorithm 1 [18] to decomposes a large order-N tensor, $\boldsymbol{\mathcal{X}}$, into N smaller order-3 core tensors, $\boldsymbol{\mathcal{G}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$ for $n = 1, \ldots, N$, such that

$$\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{G}}^{(1)} \times_3^1 \cdots \times_3^1 \boldsymbol{\mathcal{G}}^{(N)} \tag{2.6}$$

The set $\{R_0, \ldots, R_N\}$, where $R_0 = R_N = 1$, represents the TT-rank of the TTD. The $R_0 = R_N = 1$ condition allows the first and the last core tensors to be treated as matrices, $\boldsymbol{\mathcal{G}}^{(1)} \in \mathbb{R}^{1 \times I_1 \times R_1} = \boldsymbol{A} \in \mathbb{R}^{I_1 \times R_1}$ and $\boldsymbol{\mathcal{G}}^{(N)} \in \mathbb{R}^{R_{N-1} \times I_N \times 1} = \boldsymbol{B} \in \mathbb{R}^{R_{N-1} \times I_N}$. By defining $\boldsymbol{\mathcal{G}}^{(n)}_{:,i_n,:}$ as the i_n-th horizontal slice of tensor $\boldsymbol{\mathcal{G}}^{(n)}$, the MPS assumes the element-wise form as $\boldsymbol{\mathcal{X}}_{i_1,\ldots,i_N} = \boldsymbol{\mathcal{G}}^{(1)}_{:,i_1,:} \cdots \boldsymbol{\mathcal{G}}^{(N)}_{:,i_N,:}$.

Algorithm 1: TT-SVD

Input : Order-N data tensor, $\boldsymbol{\mathcal{X}}$, and approximation accuracy, ϵ
Output : Core tensors, $\boldsymbol{\mathcal{G}}^{(1)}, \ldots, \boldsymbol{\mathcal{G}}^{(N)}$, approximating $\boldsymbol{\mathcal{X}}$

Initialize cores, $\boldsymbol{\mathcal{G}}^{(1)}, \ldots, \boldsymbol{\mathcal{G}}^{(N)}$, and $R_0 = 1$
Compute truncation parameter $\delta = \frac{\epsilon}{\sqrt{N-1}} ||\boldsymbol{\mathcal{X}}||_F$
$\boldsymbol{Z} \leftarrow \boldsymbol{X}_{\{1\}}$
for $n = 1$ *to* $N - 1$ **do**
    Compute δ-truncated SVD:
    $\boldsymbol{Z} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V} + \boldsymbol{E}$, s.t. $||\boldsymbol{E}||_F \leq \delta; \boldsymbol{U} \in \mathbb{R}^{R_{n-1} I_n \times R_n}$
    $\boldsymbol{\mathcal{G}}^{(n)} \leftarrow \texttt{reshape}\,(\boldsymbol{U}, [R_{n-1}, I_n, R_n])$
    $\boldsymbol{Z} \leftarrow \texttt{reshape}\left(\boldsymbol{S}\boldsymbol{V}^T, [R_n I_{n+1}, \prod_{n'=n+2}^{N} I_{n'}]\right)$
$\boldsymbol{\mathcal{G}}^{(N)} \leftarrow \texttt{reshape}\,(\boldsymbol{Z}, [R_{N-1}, I_N, 1])$
return $\boldsymbol{\mathcal{G}}^{(1)}, \boldsymbol{\mathcal{G}}^{(2)}, \ldots, \boldsymbol{\mathcal{G}}^{(N)}$

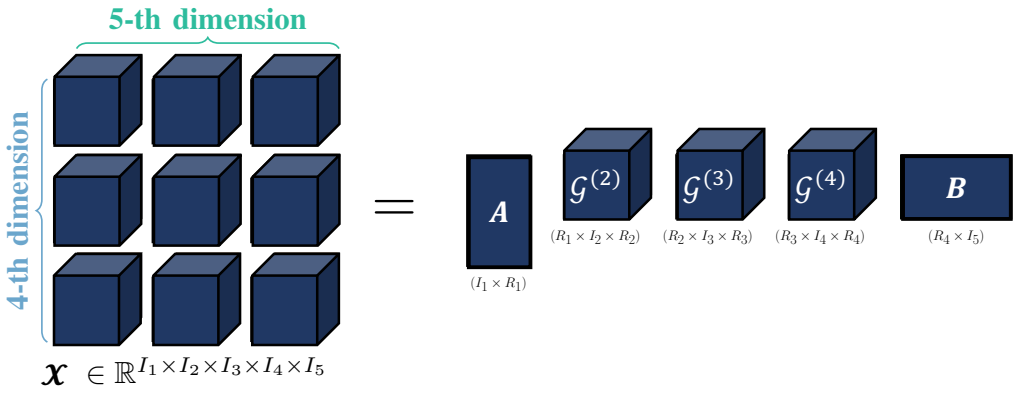


Figure 2.5: A TT type TN decomposition of an order-5 tensor, $\boldsymbol{\mathcal{X}}$, into three order-3 small-scale cores, $\boldsymbol{\mathcal{G}}^{(2)}, \boldsymbol{\mathcal{G}}^{(3)}, \boldsymbol{\mathcal{G}}^{(4)}$, and two order-2 factor matrices, $\boldsymbol{A} = \boldsymbol{\mathcal{G}}^{(1)}$ and $\boldsymbol{B} = \boldsymbol{\mathcal{G}}^{(5)}$.

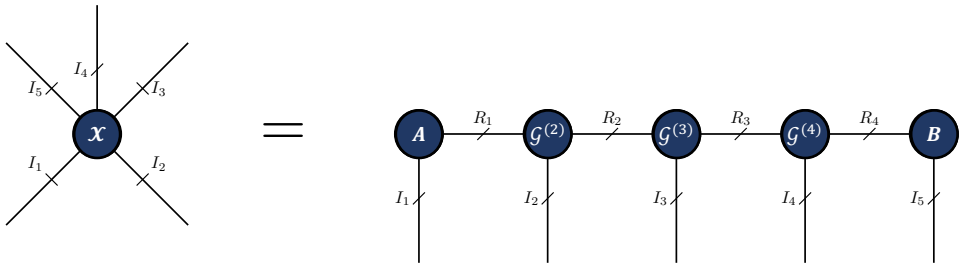


Figure 2.6: Graphical representation of TTD of an order-5 tensor, $\boldsymbol{\mathcal{X}}$ in TN notation.

Remark 1 *The MPS definition of TTD reduces the space complexity (i.e., the number of parameters) of the orignal tensor from an exponential* $\prod_{n=1}^{N} I_n$ *to a linear* $\sum_{n=1}^{N} I_n R_{n-1} R_n$ *in the dimension* I_n*, which is highly efficient for small values of* I_n*,* J_n*, and* R_n*. For* $I_n = R_n = 2$*, TTD maximizes the compression ratio of* $\frac{\prod_{n=1}^{N} I_n}{\sum_{n=1}^{N} I_n R_{n-1} R_n}$ *and is referred to as the Quantized TTD (QTTD).*

Similarly, the Matrix Product Operator (MPO) definition of the TTD [49], decomposes a large order-$2N$ tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{(I_1 \times J_1) \times \cdots \times (I_N \times J_N)}$, into smaller contracting core tensors, $\boldsymbol{\mathcal{G}}^{(n)} \in \mathbb{R}^{R_{n-1} \times (I_n \times J_n) \times R_n}$, as

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}}^{(1)} \times_4^1 \cdots \times_4^1 \boldsymbol{\mathcal{G}}^{(N)} \tag{2.7}$$

The set $\{R_0, \ldots, R_N\}$, where $R_0 = R_N = 1$, represents the TT-rank of the MPO TTD. By defining $\boldsymbol{\mathcal{G}}^{(n)}_{:,(i_n,j_n),:}$, $i_n = 1, \ldots, I_N$ and $j_n = 1, \ldots, J_N$ as the (i_n, j_n)-th slice of tensor $\boldsymbol{\mathcal{G}}^{(n)}$, the MPO assumes the element-wise form as $\boldsymbol{\mathcal{X}}_{(i_1,j_1),\ldots,(i_N,j_N)} = \boldsymbol{\mathcal{G}}^{(1)}_{:,(i_1,j_1),:} \cdots \boldsymbol{\mathcal{G}}^{(N)}_{:,(i_N,j_N),:}$.

Remark 2 *The MPO definition of TTD reduces the space complexity of the orignal tensor from an exponential* $\prod_{n=1}^{N} I_n J_n$ *to a linear* $\sum_{n=1}^{N} I_n J_n R_{n-1} R_n$ *in the dimensions* I_n *and* J_n*. For* $I_n = J_n = R_n = 2$*, TTD maximizes the compression ratio of* $\frac{\prod_{n=1}^{N} I_n J_n}{\sum_{n=1}^{N} I_n J_n R_{n-1} R_n}$.

Canonical Polyadic Decomposition: CPD [44] of rank, R, decomposes an order-N tensor, $\boldsymbol{\mathcal{W}}$, into a sum of outer products of N vectors, and is given by

$$\boldsymbol{\mathcal{W}} = \sum_{r=1}^{R} \boldsymbol{u}_r^{(1)} \circ \cdots \circ \boldsymbol{u}_r^{(N)}, \tag{2.8}$$

where $\boldsymbol{u}_r^{(n)} \in \mathbb{R}^{I_n}$ denotes the r-th mode-n factor vector, as illustrated diagrammatically in Figure 2.7.

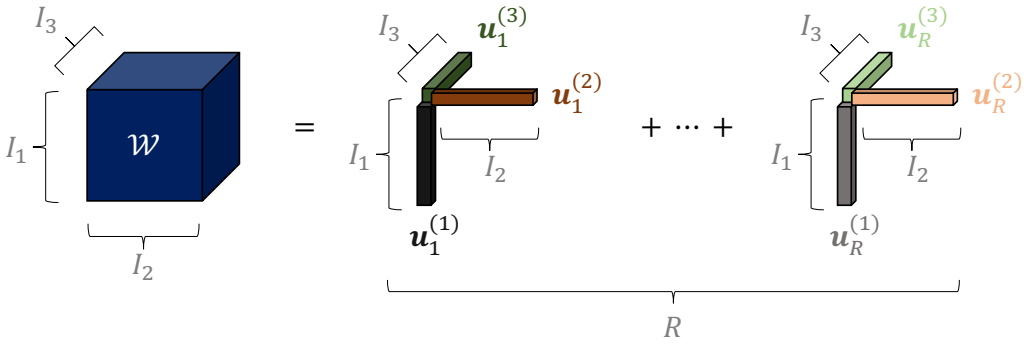


Figure 2.7: Visualization of the CPD for an order-3 tensor, $\boldsymbol{\mathcal{W}}$, according to (2.8). The sizes of tensor modes, I_1, I_2, and I_3, are labelled in gray.

Alternatively, for each mode-n, we can stack the vectors, $\boldsymbol{u}_r^{(n)}$ for $r = 1, \ldots, R$, to form the factor matrix, $\boldsymbol{U}^{(n)} \in \mathbb{R}^{I_n \times R}$, as $\boldsymbol{U}^{(n)} = [\boldsymbol{u}_1^{(n)}, \ldots, \boldsymbol{u}_R^{(n)}]$. This allows us to express the CPD as a series of tensor contractions, given by

$$\boldsymbol{\mathcal{W}} = \boldsymbol{\mathcal{I}} \times_1^1 \boldsymbol{U}^{(1)} \times_2^1 \cdots \times_N^1 \boldsymbol{U}^{(N)}, \tag{2.9}$$

as illustrated in Figure 2.8 and Figure 2.9 in standard matrix notation and TN notation. In the CPD format, the mode-n matricization of a tensor can be expressed as a series of Khatri-Rao products of the factors matrices, as

$$\begin{aligned} \boldsymbol{W}_{\{n\}} &= \boldsymbol{U}^{(n)} \left(\boldsymbol{U}^{(N)} \odot \cdots \odot \boldsymbol{U}^{(n+1)} \odot \boldsymbol{U}^{(n-1)} \odot \boldsymbol{U}^{(1)}\right)^T \\ &= \boldsymbol{U}^{(n)} \left(\bigodot_{i=N, i\neq n}^{1} \boldsymbol{U}^{(i)} \right)^T \\ &= \boldsymbol{U}^{(n)} \boldsymbol{U}^{(-n)^T}, \end{aligned} \tag{2.10}$$

where $\boldsymbol{U}^{(-n)} = \left(\bigodot_{i=N, i\neq n}^{1} \boldsymbol{U}^{(i)}\right)$. Similarly, the vectorization of a tensor in its CPD format can be expressed as

$$\begin{aligned} \text{vec}(\boldsymbol{\mathcal{W}}) &= \left(\boldsymbol{U}^{(N)} \odot \cdots \odot \boldsymbol{U}^{(1)}\right) \mathbf{1} \\ &= \left(\bigodot_{i=N}^{1} \boldsymbol{U}^{(i)} \right) \mathbf{1}, \end{aligned} \tag{2.11}$$

where $\mathbf{1}$ denotes a vector of unities.

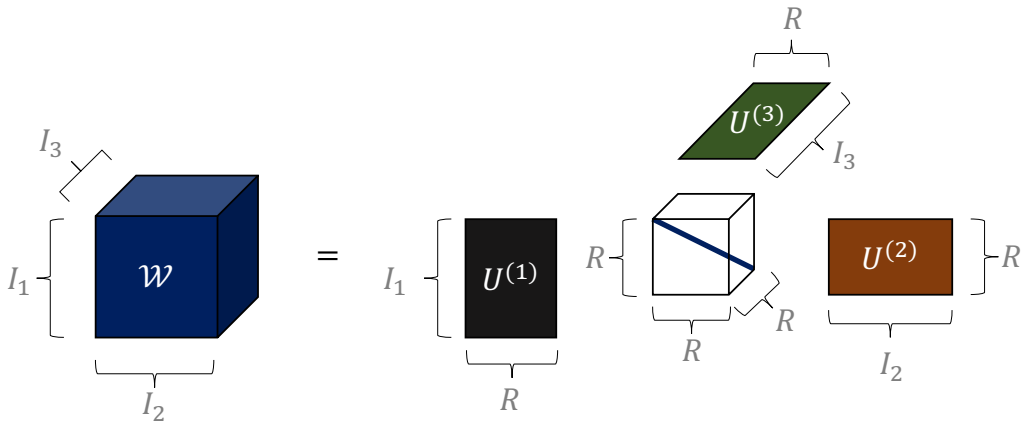


Figure 2.8: Alternative visualization of the CPD for an order-3 tensor, $\mathcal{W}$, according to (2.9) in standard notation. The sizes of tensor modes, I_1, I_2, and I_3, are labelled in gray.

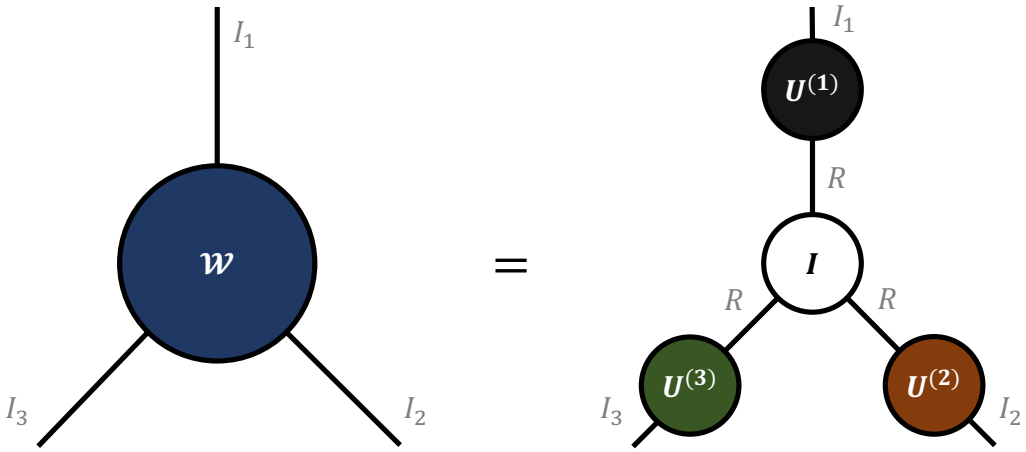


Figure 2.9: Alternative visualization of the CPD for an order-3 tensor, $\mathcal{W}$, according to (2.9) in TN notation. The sizes of tensor modes, I_1, I_2, and I_3, are labelled in gray.

In practice, given an order-N tensor, $\boldsymbol{\mathcal{W}}$, the estimation of the factor matrices, $\boldsymbol{U}^{(1)}, \ldots, \boldsymbol{U}^{(N)}$, are done via the Alternating Least Squares (ALS) algorithm described in Algorithm 2 [18].

Algorithm 2: CPD via ALS

Input : Tensor, $\boldsymbol{\mathcal{W}}$, rank R, and approximation accuracy, δ
Output : Factor matrices, $\boldsymbol{U}^{(1)}, \ldots, \boldsymbol{U}^{(N)}$
Initialize factor matrices, $\boldsymbol{U}^{(1)}, \ldots, \boldsymbol{U}^{(N)}$
Compute approximation accuracy $\epsilon = ||\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{I}} \times_1^1 \boldsymbol{U}^{(1)} \times_2^1 \cdots \times_N^1 \boldsymbol{U}^{(N)}||_F$
while $\delta < \epsilon$ **do**
  for $n = 1$ *to* N **do**
    $\boldsymbol{U}^{(n)} \leftarrow \boldsymbol{W}_{\{n\}} \left(\odot_{i=N, i\neq n}^1 \boldsymbol{U}^{(i)}\right) \left(\circledast_{i=N, i\neq n}^1 \boldsymbol{U}^{(i)^T} \boldsymbol{U}^{(i)}\right)$
  $\epsilon \leftarrow ||\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{I}} \times_1^1 \boldsymbol{U}^{(1)} \times_2^1 \cdots \times_N^1 \boldsymbol{U}^{(N)}||_F$
return $\boldsymbol{U}^{(1)}, \ldots, \boldsymbol{U}^{(N)}$

2.2 Preliminaries on Graphs

Graph: A graph, $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, is defined by a set of I vertices (or nodes) $v_i \subset \mathcal{V}$ for $i = 1, \ldots, I$, and a set of edges connecting the ${i_1}^{th}$ and ${i_2}^{th}$ vertex, $(v_{i_1}, v_{i_2}) \in \mathcal{E}$, for $i_1 = 1, \ldots, I_1$ and $i_2 = 1, \ldots, I_2$, where $I_1 = I_2 = I$.

Neighbourhood: The neighbourhood of the i-th node, $\mathcal{N}_i$, is the set of nodes that are connected with the i-th node. The number of nodes in the neighbourhood is denoted by $|\mathcal{N}_i|$. A graph where every node has exactly K neighbours is also referred to as K-regular.

Graph Shift Operators: A graph can be described by its Graph Shift Operator (GSO), $\boldsymbol{S} \in \mathbb{R}^{I_1 \times I_2}$, a square matrix such that the entry, $\boldsymbol{S}_{i_1,i_2} \in \mathbb{R}$, encodes how the information should shift from the i_2-th node to the i_1-th node.

Graph Adjacency Matrix: Special cases of GSOs include the graph adjacency matrix, $\boldsymbol{A} \in \mathbb{R}^{I_1 \times I_2}$, where the entry of the GSO, $\boldsymbol{A}_{i_1,i_2} = 1$, if there exists an edge that connects to i_1-th node from the i_2-th node, and zero otherwise. Similarly, the weighted graph adjacency matrix encodes the entry, $\boldsymbol{A}_{i_1,i_2} \in \mathbb{R}$, to reflect the corresponding edge weight. A graph is said to be undirected if $\boldsymbol{A} = \boldsymbol{A}^T$, and directed otherwise.

Graph Degree Matrix: The graph degree matrix, $\boldsymbol{D} = \text{degree}\,(\boldsymbol{A}) \in \mathbb{R}^{I_1 \times I_2}$, is a diagonal matrix with entries $\boldsymbol{D}_{i_1,i_1} = \sum_{i_2} \boldsymbol{A}_{i_1,i_2}$, where $\boldsymbol{A}_{i_1,i_2}$ denotes the (i_1, i_2)-th entry of the corresponding graph adjacency matrix.

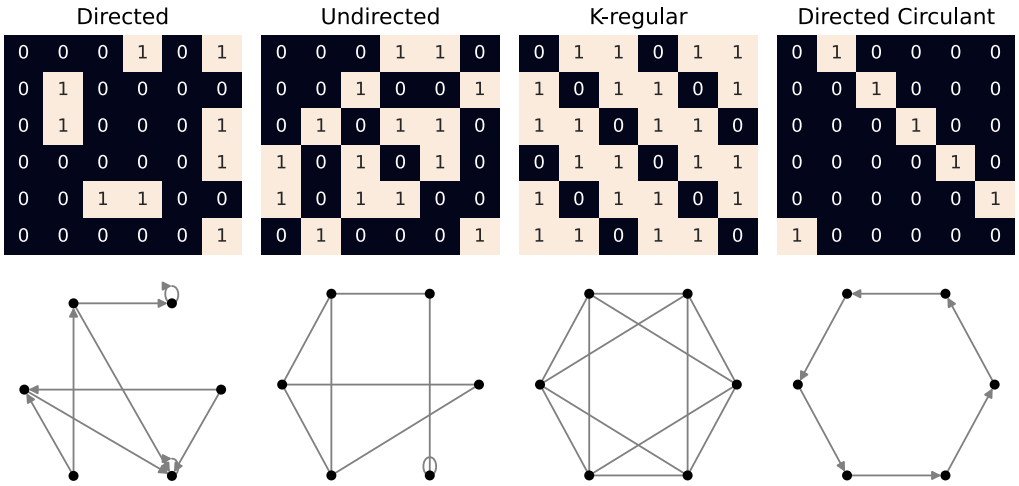


Figure 2.10: The graph adjacency matrix (top panels) and the corresponding graph topology (bottoms) for (i) a directed graph, (ii) an undirected graph, (iii) a K-regular graph with $K = 4$, and (iv) a DCG with offset $k = 1$.

Graph Laplacian Matrix: The graph Laplacian matrix is a GSO defined as $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A} \in \mathbb{R}^{I_1 \times I_2}$, where $\boldsymbol{A}$ is the graph adjacency matrix, and $\boldsymbol{D}$ is the diagonal degree matrix.

Normalized GSOs: A GSO, $\boldsymbol{S}$, is said to be normalized if it is scaled by the corresponding degree matrix as $\tilde{\boldsymbol{S}} = \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{S} \boldsymbol{D}^{-\frac{1}{2}}$, where $\boldsymbol{D}$ is the corresponding diagonal degree matrix.

Graph Inference: Given a set of vertices and their descriptive features, an unknown GSO (i.e., an unknown graph topology) can be inferred through the use of a pairwise similarity function, $f(\cdot)$, such that the edge weights are estimated as $\boldsymbol{S}_{i_1,i_2} = f\left(\boldsymbol{x}^{(i_1)}, \boldsymbol{x}^{(i_2)}\right)$, where $\boldsymbol{x}^{(i_1)}$ and $\boldsymbol{x}^{(i_2)}$ are the features associated with the i_1-th and i_2-th nodes.

Circulant Graphs: A Directed Circulant Graph (DCG) with offsets $k = 1, \ldots, K$ [62] is a graph characterized by a circulant graph adjacency matrix (i.e., a diagonal constant matrix) such that, $\boldsymbol{A}_{j,(j+k-1)\%I} \neq 0$ for $j = 1, \ldots, J$ and $k = 1, \ldots, K$, where $\%$ denotes the modulo operation.

Graph Signals: A graph signal, $\boldsymbol{x} \in \mathbb{R}^I$, is a vector of size I that associates a scalar value to each of the I vertices in the graph (e.g., the age of users in a social network graph). Given a set of J graph signals, these can be stacked as column vectors to form the graph signal matrix, $\boldsymbol{X} \in \mathbb{R}^{I \times J}$.

Graph Signal Smoothness: The smoothness of a graph signal, $\boldsymbol{x} \in \mathbb{R}^I$, with respect to the underlying graph topology can then be quantified as *smoothness* $= \boldsymbol{x}^T \boldsymbol{L} \boldsymbol{x}$, which measures the change in signal value between neighbouring (i.e., connected) nodes [34]. Therefore, a graph signal is said to be smooth if it does not change abruptly across connected nodes, as illustrated in Figure 2.11. For a given set of R different graph signals, these can be stacked as a set of column vectors into a matrix, $\boldsymbol{X} \in \mathbb{R}^{I \times R}$. The smoothness can then be computed as

$$\textit{smoothness} = \operatorname{tr}\left(\boldsymbol{X}^T \boldsymbol{L} \boldsymbol{X}\right). \tag{2.12}$$

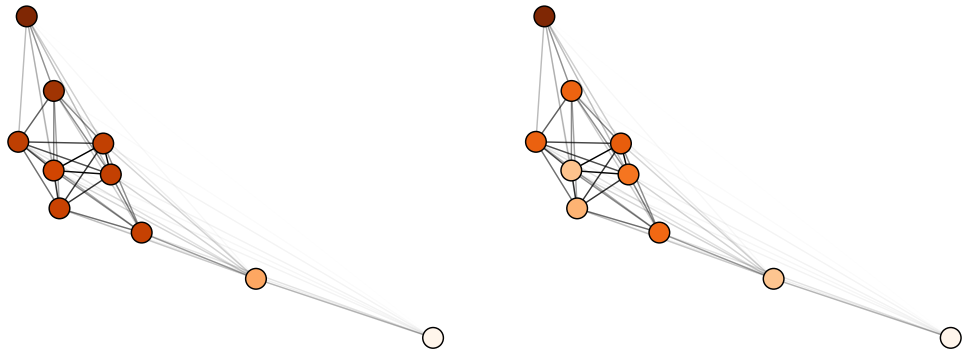


Figure 2.11: Smooth (left) and non-smooth (right) graph signals. The vertices of the illustrated graphs are coloured proportionally to the graph signal values, while the edges are also coloured proportionally to their edge weight. A graph signal is considered smooth if it varies slowly over closely connected neighbouring vertices.

Processing Graph Signals via GSOs: The matrix-by-vector product between a GSO, $\boldsymbol{S} \in \mathbb{R}^{I \times I}$, and a graph signal, $\boldsymbol{x} \in \mathbb{R}^I$, generates another graph signal, $\boldsymbol{y} = \boldsymbol{S}\boldsymbol{x} \in \mathbb{R}^I$, which can have several well-defined physical meanings depending on the choice of the GSO [63]. For instance, a graph adjacency matrix based GSO performs a neighbourhood aggregation operation where the graph signal at the i_1-th vertex, $\boldsymbol{y}_{i_1}$, results from the sum of graph signals from its neighbouring (i.e., connected) vertices, $\boldsymbol{y}_{i_1} = \sum_{i_2} \boldsymbol{S}_{i_1,i_2} \boldsymbol{x}_{i_2}$. This can be naturally extended to graph signal matrices for J graph signals as $\boldsymbol{Y} = \boldsymbol{S}\boldsymbol{X} \in \mathbb{R}^{I \times J}$.

Graph Fourier Transform: If the underlying graph is undirected, then the adjacency matrix admits an eigen-decomposition as $\boldsymbol{A} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^T$, where $\boldsymbol{\Phi}$ is the matrix of orthonormal eigenvectors, and $\boldsymbol{\Lambda}$ is the diagonal matrix of real-valued eigenvalues. In spectral graph theory, the eigenvectors (the columns of $\boldsymbol{\Phi}$) are regarded as the graph Fourier bases of a given graph domain, whereby the corresponding eigenvalues in $\boldsymbol{\Lambda}$ represent the associated frequencies. More specifically, given a graph signal, $\boldsymbol{x}$, its graph Fourier transform can be computed as $\hat{\boldsymbol{x}} = \boldsymbol{\Phi}^T \boldsymbol{x}$, while the inverse Fourier transform can be computed as $\boldsymbol{x} = \boldsymbol{\Phi}\hat{\boldsymbol{x}}$.

Remark 3 *The Graph Fourier Transform can be defined both in terms of the adjacency matrix and the Laplacian matrix, albeit with the opposite eigenvalue ordering. For instance, the smoothest eigenvector (lowest frequency) is associated with the largest eigenvalue for the adjacency matrix case, but it is associated with the smallest eigenvalue in the Laplacian matrix case [34].*

Spectral Graph Filters: A spectral graph filter is a diagonal matrix of spectral multipliers, $\mathbf{\Gamma} \in \mathbb{R}^{I \times I}$, which can be applied to extract features from a graph signal, $\boldsymbol{x} \in \mathbb{R}^I$, via the spectral convolution operation, $\boldsymbol{y} = \mathbf{\Phi}\mathbf{\Gamma}\mathbf{\Phi}^T\boldsymbol{x} \in \mathbb{R}^I$ [33]. This follows from the convolution theorem where the spatial convolution of two signals correspond to their product in the Fourier domain.

Spatial Graph Filters: The GSO can be used to filter a set of J signals on a graph with I vertices, $\boldsymbol{X} \in \mathbb{R}^{I \times J}$, as $\boldsymbol{Y} = \sum_{k=0}^{K-1} \boldsymbol{\alpha}_k \boldsymbol{A}^k \boldsymbol{X}$, where K is the order of the polynomial GSO filter and $\boldsymbol{\alpha}_k$ is the filter constant associated with the k-th power. Such a spatial graph filter represents a linear combination of vertex-shifted graph signals, which captures graph information at a local level.

Part I

Tensor Learning Methods

Chapter 3

Tensor Contraction via Tensor-Train Decomposition

A key tool underpinning multi-linear manipulation of tensors and TNs is the standard TCP, which is at the heart of all ML algorithms discussed in this thesis. However, depending on the dimensionality of the underlying tensors, the TCP also comes at the price of high computational complexity in tensor manipulation. In this chapter, we resort to diagrammatic TN manipulation to calculate such products in an efficient and computationally tractable manner, by making use of TTD. This has rendered the underlying concepts easy to perceive, thereby enhancing intuition of the associated underlying operations, while preserving mathematical rigour. In addition to bypassing the cumbersome mathematical multi-linear expressions, the proposed TTCP model is shown to accelerate significantly the underlying computational operations, as it is independent of tensor order and linear in the tensor dimension, as opposed to performing the full computations through the standard approach (exponential in tensor order).

3.1 Introduction

Tensors and TDs have recently received tremendous attention in the data analytics community, due to their ability to mitigate the curse of dimensionality associated with modern large-dimensional big data [18, 18]. Indeed, TDs allow for data volume (e.g. the parameter complexity) to be reduced from scaling exponentially to scaling linearly in the tensor dimensions, which facilitates applications in areas including the compression and interpretability of Neural Networks [30, 18], multi-modal learning [18], and completion of Knowledge Graphs [31, 32]. At the heart of TD techniques is the TCP, an operator used for representing even the most unmanageable higher order tensors through a set of small-scale *core tensors* that are inter-connected via TCP operations [18].

Despite success of TDs, a major obstacle towards a more widespread use of tensors in research and applications is that even relatively compact multilinear operations, such as the Tucker (TKD) and the TTDs, tend to be awkward to implement. This is due to the overwhelming amount of indexing required for higher order tensors,

and the associated cumbersome mathematical formulae. For these reasons, TDs may not immediately appeal to data analytics communities in the same way as their "flat-view" matrix counterparts; in addition, multi-linear operations become progressively harder to visualize with an increase in data dimensionality and tensor order. These issues are rapidly becoming even more acute given that the range of "big data" applications is growing at a fast rate.

As a remedy to these computational and conceptual obstacles, diagrammatic (graphical) approaches have been developed in quantum physics and quantum chemistry [18, 64] to illustrate and implement mathematical operations for tensors. For example, contractions of arbitrary TNs on graphs have been considered in the context of spin glass energy and quantum circuits in [65]. In this way, tensors and by continuity also TNs, are represented as a connected graph of nodes (tensors) and edges (their modes), in a way that resembles mathematical graphs. Such graphical tools provide an intuitive and concise representation of multilinear operations, while at the same time maintaining the underlying mathematical rigour, but are only recently being adopted in signal processing and data analytics communities.

The advantages of graphical representation of tensors are illustrated in Figure 2.4, which shows that a tensor of any order, including scalars (order-0), vectors (order-1), matrices (order-2) and higher order tensors, can be represented by circles (or nodes) within a TN diagram. The edges of a given tensor represent the tensor modes (dimensions), and, conventionally, they are labelled from the "top" of a node (North) in an anti-clockwise direction; the edge labels designate the dimensionality of the mode, while the number of edges connected to the node designates the order of the tensor. A *contraction mode* is an edge that connects two tensor modes of equal size, while an edge which is not connected to another tensor represents a *physical mode* of the underlying tensors, or a TN.

Remark 4 *The TCP is at the core of the manipulation of large tensors, however, its computation comes at an exponentially increasing time complexity (i.e., the number of basic operations such as addition and multiplication) of $\mathcal{O}(I^{2N-1})$, with N as the tensor order and I as the modal dimension. To put this into context, for an order-5 tensor with all modal dimensions of $I = 10$, the computation of TCP would require a whopping $\mathcal{O}(10^9)$ operations; this is prohibitive to real-world applications based on tensors.*

The aim of this chapter is therefore to leverage on the power and flexibility of the graphical TN representations in order to:

1. Develop a TCP algorithm via TTD type of TN, termed the TTCP; this is achieved using purely the intuitive diagrammatic representation of TNs, and without having to resort to awkward multi-indexing, and cumbersome and non-obvious mathematical expressions.

2. Reduce the time complexity of TCP from the original exponential one in the tensor order, $\mathcal{O}(I^{2N-1})$, to $\mathcal{O}(IR^2)$ and independent of the tensor order in TTCP, if the tensors are stored in the TT format.

It is our hope that the intuitive and mathematics-free approach presented in this chapter will help both better understanding and more efficient implementation of

tensor operations, together with demystifying their multilinear algebra for wider data analytics communities, and promoting their more widespread adoption among researchers and practitioners.

3.2 Contraction in the Tensor-Train Format

Tensor representation through nodes and edges is immensely powerful, as it offers both flexibility in tensor manipulation and versatility in their mathematical operations. Indeed, this makes it possible to e.g. split links, contract cores, rotate sub-networks, and merge TNs by contracting specific cores, when an original tensor is treated as a graph. We next show that *the topology of a TN is sufficient to obtain insights which would otherwise be non-obvious and difficult to gain*. Owing to its fundamental nature, our focus is on the TCP. We next show that when performed graphically through a TT-type of TN, the TCP becomes intuitive, straightforward to implement, and computationally efficient, whereby each core conveys information regarding the corresponding original tensor mode.

Consider the MPS definition of TTD discussed in 2.1. Given that the first and last cores in a TT are always matrices, the question arises whether it is possible to exploit such an arrangement to reduce an $\times_n^m$ contraction product to a simple matrix multiplication.

Example 2 *For illustration, consider a $\times_3^2$ contraction between a 4th-order tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4}$ and a 3rd-order tensor, $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$, where $I_3 = J_2$. This results in a tensor $\boldsymbol{\mathcal{Z}}$, given by*

$$\boldsymbol{\mathcal{X}} \times_3^2 \boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{Z}} \in \mathbb{R}^{I_1 \times I_2 \times I_4 \times J_1 \times J_3} \tag{3.1}$$

The tensors $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4}$ and $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$ can also have their indices permuted to yield $\boldsymbol{\mathcal{X}}^{\rho} \in \mathbb{R}^{I_3 \times I_1 \times I_2 \times I_4}$ and $\boldsymbol{\mathcal{Y}}^{\rho} \in \mathbb{R}^{J_2 \times J_1 \times J_3}$, such that the contracting modes, I_3 for $\boldsymbol{\mathcal{X}}$, and J_2 for $\boldsymbol{\mathcal{Y}}$, become the first modes of $\boldsymbol{\mathcal{X}}^{\rho}$ and $\boldsymbol{\mathcal{Y}}^{\rho}$ (the permutation operation only requires $\mathcal{O}(1)$ complexity). Upon performing the TT decomposition on $\boldsymbol{\mathcal{X}}^{\rho}$ and $\boldsymbol{\mathcal{Y}}^{\rho}$ respectively, this yields the constituent cores

$$\begin{aligned} \boldsymbol{\mathcal{G}}_x^{(n)} &\in \mathbb{R}^{R_{n-1} \times I_n \times R_n}, \;\; n = 1, 2, 3, 4 \\ \boldsymbol{\mathcal{G}}_y^{(m)} &\in \mathbb{R}^{P_{m-1} \times J_m \times P_m}, \;\; m = 1, 2, 3 \\ P_0 &= P_3 = 1 \\ R_0 &= R_4 = 1 \end{aligned} \tag{3.2}$$

This implies that, since $I_3 = J_2$, the desired contraction in (3.1) *can be obtained as a simple matrix product $\boldsymbol{\mathcal{G}}_x^{(1)} = \boldsymbol{A}_x \in \mathbb{R}^{I_3 \times R_1}$ and $\boldsymbol{\mathcal{G}}_y^{(1)} = \boldsymbol{A}_y \in \mathbb{R}^{J_2 \times P_1}$, in the form*

$$\boldsymbol{A}_x^T \boldsymbol{A}_y = \boldsymbol{K} \in \mathbb{R}^{R_1 \times P_1} \tag{3.3}$$

The intuition in Example 2 is at the centre of the proposed TTCP, a method to perform contractions between tensors that are represented in the TT-format. This is illustrated in Figure 3.1, where the cores $\boldsymbol{A}_x$ and $\boldsymbol{A}_y$ of the tensors $\boldsymbol{\mathcal{X}}^{\rho}$ and $\boldsymbol{\mathcal{Y}}^{\rho}$ are linked by a standard matrix multiplication, $\boldsymbol{A}_x^T \boldsymbol{A}_y$, connecting the two TT

representations of $\boldsymbol{\mathcal{X}}^{\rho}$ and $\boldsymbol{\mathcal{Y}}^{\rho}$. This results in a TT (middle panels in Figure 3.1), whereby the link introduced by the contraction product is consolidated into the matrix, $\boldsymbol{K} \in \mathbb{R}^{R_1 \times P_1}$. Thus, the task of reconstructing the tensor represented by the union of the two TTs yields $\boldsymbol{\mathcal{Z}}^{\rho} \in \mathbb{R}^{I_4 \times I_2 \times I_1 \times J_1 \times J_3}$, which is a permuted version of the tensor $\boldsymbol{\mathcal{Z}}$ in (3.1). To obtain the latter, it is sufficient to permute $\boldsymbol{\mathcal{Z}}^{\rho}$ such that the resulting dimensions follow the natural anti-clockwise order as in the original contraction. The proposed TTCP procedure is summarised in Algorithm 3.

Remark 5 *The TTCP in Figure 3.1 is completely mathematics-free as it stems purely from diagrammatic tensor-manipulation foundations, and makes a full use of the topology of the graphical TT representations of tensors [49].*

Algorithm 3: TTCP

Input : $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$, n, m

Output : $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{X}} \times_n^m \boldsymbol{\mathcal{Y}}$

1 Perform permutation of an original $\boldsymbol{\mathcal{X}} \rightarrow \boldsymbol{\mathcal{X}}^{\rho}$ such that

$$\boldsymbol{\mathcal{X}}^{\rho} \in \mathbb{R}^{I_n \times I_1 \times \cdots \times I_{n-1} \times I_{n+1} \cdots \times I_N}$$

2 Represent $\boldsymbol{\mathcal{X}}^{\rho}$ in the TT format using TT-SVD

$$\boldsymbol{\mathcal{X}}^{\rho} = \boldsymbol{A}_x \times_2^1 \boldsymbol{\mathcal{G}}_x^{(2)} \times_3^1 \cdots \times_3^1 \boldsymbol{\mathcal{G}}_x^{(N-1)} \times_3^1 \boldsymbol{B}_x$$

3 Repeat steps 1 and 2 for $\boldsymbol{\mathcal{Y}}$ to obtain

$$\boldsymbol{\mathcal{Y}}^{\rho} = \boldsymbol{A}_y \times_2^1 \boldsymbol{\mathcal{G}}_y^{(2)} \times_3^1 \cdots \times_3^1 \boldsymbol{\mathcal{G}}_y^{(M-1)} \times_3^1 \boldsymbol{B}_y$$

4 Perform contraction $\times_n^m$ through matrix multiplication

$$\boldsymbol{K} = \boldsymbol{A}_x^T \boldsymbol{A}_y$$

5 Obtain reconstructed version of $\boldsymbol{\mathcal{Z}}^{\rho}$

$$\begin{aligned} \boldsymbol{\mathcal{Z}}^{\rho} = & \boldsymbol{B}_x \times_2^1 \boldsymbol{\mathcal{G}}_x^{(N-1)} \times_3^1 \cdots \\ & \cdots \times_3^1 \boldsymbol{\mathcal{G}}_x^{(2)} \times_3^1 \boldsymbol{K} \times_2^1 \boldsymbol{\mathcal{G}}_y^{(2)} \times_3^1 \cdots \\ & \cdots \times_3^1 \boldsymbol{\mathcal{G}}_y^{(M-1)} \times_3^1 \boldsymbol{B}_y \end{aligned}$$

6 Perform permutation of $\boldsymbol{\mathcal{Z}}^{\rho} \rightarrow \boldsymbol{\mathcal{Z}}$ such that

$$\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N \times J_1 \times \cdots \times J_{m-1} \times J_{m+1} \times \cdots \times J_M}$$

7 **return** $\boldsymbol{\mathcal{Z}}$

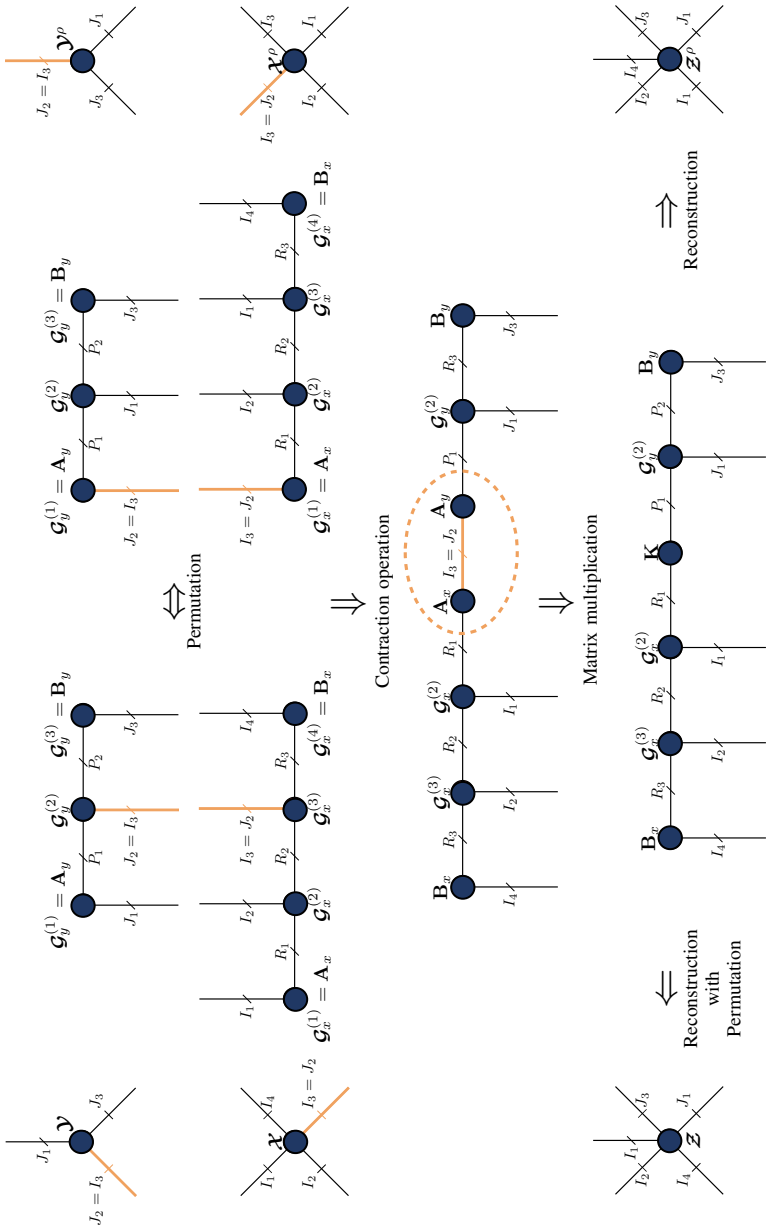


Figure 3.1: Graphical illustration of the TTCP. The contraction is performed on an order-4 tensor, $\boldsymbol{\mathcal{X}}$, and an order-3 tensor, $\boldsymbol{\mathcal{Y}}$, along their common modes $I_3 = J_2$. By exploiting the topology of the TT representations of the original tensors, $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$, the TT decompositions of the permuted versions of $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$ are obtained, allowing for the reduction of the otherwise computationally prohibitive contraction in (2.5) to a single matrix multiplication.

The benefits of the TTCP are therefore two-fold:

1. The resulting tensor $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{X}} \times_3^2 \boldsymbol{\mathcal{Y}}$ is automatically given in the TT-format, and is thus suitable for efficient distributed storage, while its cores still contain the physically meaningful modal information with respect to the individual tensors $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$.

2. It provides a computationally more efficient alternative to the *standard* definition in (2.5) of the contraction product, especially for higher order tensors.

3.3 Computational Complexity Analysis

Generally, if all tensors are stored in the TT format, then instead of performing an expensive $\times_n^m$ TCP between the tensors $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$ according to (2.5), which incurs $\mathcal{O}(I_1 \cdots I_{n-1} I_{n+1} \cdots I_N I_n J_1 \cdots J_{m-1} J_{m+1} \cdots J_M)$ operations, the proposed TTCP only requires $\mathcal{O}(R_1 I_n P_1)$ operations, where R_1 is the first TT rank of $\boldsymbol{\mathcal{X}}$ and and P_1 is the first TT rank of $\boldsymbol{\mathcal{Y}}$.

For illustration on the savings in computational complexity by the proposed TTCP, let $I_n = I$ and $J_m = I$ for $n = 1, \ldots, N$ and $m = 1, \ldots, N$, and $R_1 = P_1 = R$. Then, the number of operations required (so called *time complexity*) of the TCP reduces from the original $\mathcal{O}(I^{2N-1})$, which is *exponential in the tensor order*, N, to $\mathcal{O}(IR^2)$ in the TTCP (Step 4 of Algorithm 3), which is *independent of the tensor order*, and is highly efficient for a low TT rank, R, as illustrated in Figure 3.2. Hence, the computational advantage of TTCP over standard TCP is $\mathcal{O}(I^{2N-1})/\mathcal{O}(IR^2) \approx \mathcal{O}(I^{2N-2})$.

Notice that if the original tensors need first to be represented in the TT format, then this would incur an additional computational complexity of $\mathcal{O}(RI^N)$ [66] (steps 2 and 3 of Algorithm 3). In this case, the computational advantage of TTCP would still be of the order $\mathcal{O}(I^{N-1})$. Finally, if the resulting contraction product needs to be reconstructed from the TT-format to the full tensor form, this would then incur an additional $\mathcal{O}(I^{2N-2})$ complexity, whereby the computational advantage of TTCP remains $\mathcal{O}(I)$.

In practice, however, it is desirable to store tensors in a distributed TT-format to reduce the storage and computational overheads associated with large-scale TNs, and only reconstruct the final tensor after performing numerous contraction products. This is an ideal scenario for the proposed TTCP to provide large complexity gains over standard contractions.

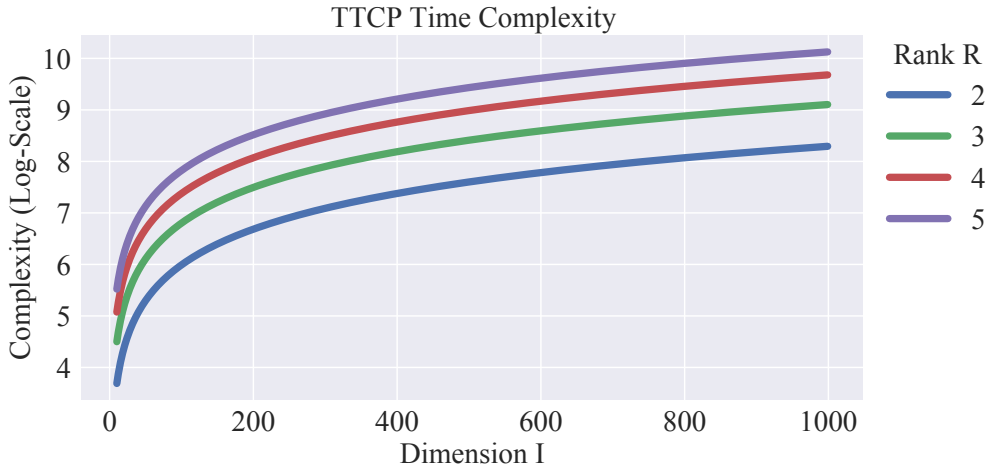


Figure 3.2: Theoretical time complexity (in log-scale) required to perform the proposed TTCP for tensors in TT-format with varying dimensions, I, and TT ranks, R.

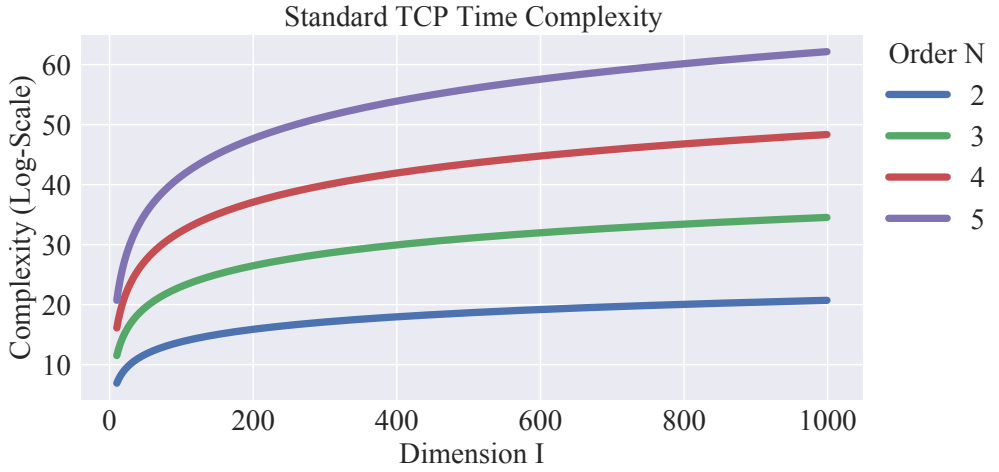


Figure 3.3: Theoretical time complexity (in log-scale) required to perform the standard contraction product for tensors with varying dimensions, I, and tensor orders, N.

3.4 Numerical Examples

The computational efficiency of TTCP is illustrated in Figure 3.4, where two tensors, $\mathcal{X}$ and $\mathcal{Y}$, are drawn from a zero-mean, unit-variance Gaussian distribution, and three different cases are considered: (i) $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{20\times20\times20}$; (ii) $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{20\times20\times20\times5}$; and (iii) $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{20\times20\times20\times5\times4}$. For each case, the contraction $\mathcal{X} \times_1^1 \mathcal{Y}$ was computed both in the original form, via (2.5), and also through the proposed TTCP algorithm. The implementations were performed using the software library HOTTBOX [67], and the computational times were recorded for both cases. Observe that the TTCP exhibits a significant computational efficiency advantage in all cases, especially for higher order tensors, such as the order-4 and order-5 tensors considered here, where expression (2.5) quickly tends to become intractable.

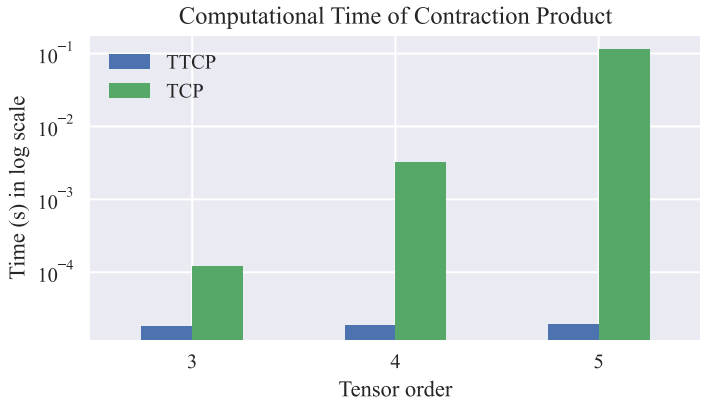


Figure 3.4: Computational time (in seconds, log-scale) of the TCP and the proposed TTCP, for order-3, order-4 and order-5 tensors.

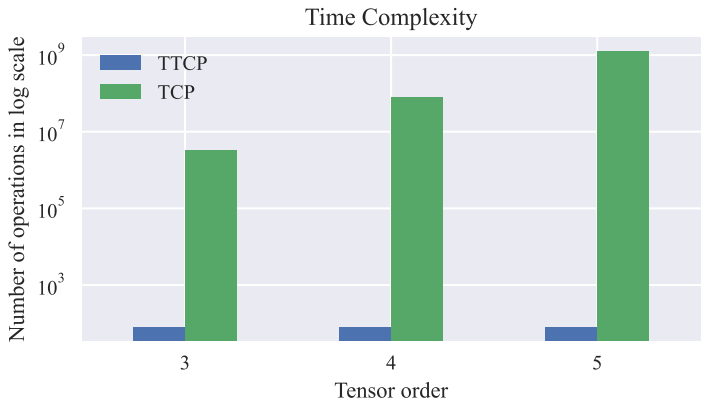


Figure 3.5: Theoretical time complexity (in number of operations, log-scale) of the TCP and the proposed TTCP corresponding to Figure 3.4.

3.5 Approximation Error

The TT representation is a low-rank approximation of the original tensor, which inevitably comes at a cost of approximation error. Generally, the higher the rank of the actual TTD, the less pronounced the approximation error. However, the determination of the exact rank for a tensor is generally NP-hard [68], which is prohibitive to providing a theoretical guarantee on the approximation error. Instead, in practice, the choice of rank is determined empirically for a given error tolerance, and it varies significantly across different datasets. As a rule of thumb, we find that a low TT-rank of $R_n = 4$ for $n = 1, \ldots, N-1$ is often sufficient to approximate more than 90% of the original TCP (measured in explained variance) in some of the most popular ML benchmark datasets, as shown in Figure 3.6.

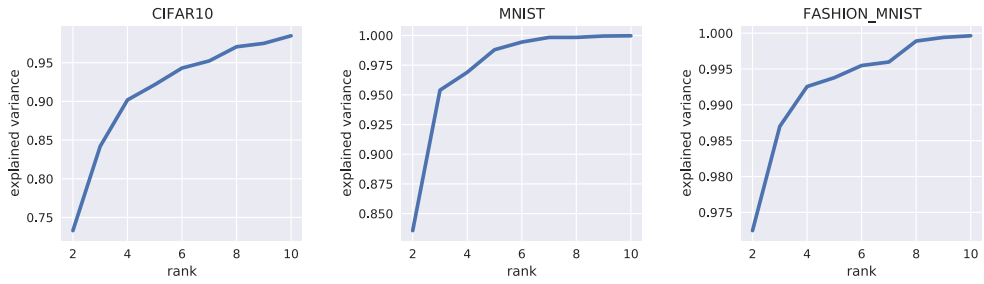


Figure 3.6: The TTCP approximation error measured in explained variance (y-axis) as a function of TT rank (x-axis) across three popular ML datasets. Note that a TTCP with a low rank of $R_n = 4, n = 1, \ldots, N-1$ is often enough to approximate the original contraction product by more than 90%.

3.6 Summary

The ever expanding trends in both the generated data volume and range of big data applications have highlighted the necessity for more versatile analysis tools that offer greater opportunities for algorithmic developments, together with the urge for computationally faster operations than with the standard "flat-view" matrix approaches. Tensors, or multi-way arrays, provide such an algebraic framework which is naturally suited to data of large volume, diversity, and veracity. Indeed, the associated TDs have demonstrated, in manifold areas, their potential in breaking the curse of dimensionality associated with traditional matrix methods, whereby a necessary exponential increase in data volume leads to adverse effects on computational complexity.

A key tool underpinning multi-linear manipulation of tensors and TNs is the standard TCP. However, depending on the dimensionality of the underlying tensors, the TCP also comes at the price of an exponential increase in computational complexity in tensor order. To address this issue, we employ diagrammatic TN manipulation in order to calculate such products in an efficient, maths-free, and computationally

tractable manner. This has been achieved by making use of TTD, which has rendered the underlying concepts straightforward to interpret and implement, thereby enhancing intuition of the associated multilinear operations, while preserving mathematical rigour. In addition to completely bypassing the cumbersome mathematical multi-linear expressions and thus eliminating the main source of errors in tensor manipulation, the introduced TTCP model has been shown to accelerate significantly the underlying computational operations, as it is independent of the tensor order for tensors stored in the distributed TT format.

It is our hope that this perspective will help demystify tensor approaches for multidisciplinary Data Analysis communities, and at the same time serve as a platform to explore numerous opportunities in manifold applications of our ever growing data-hungry world. We also hope that the conceptual insights through TN graphs will encourage researchers and practitioners to explore a whole host of otherwise computationally prohibitive modern applications based on big data and tensors.

Chapter 4

Tensor-Train Recurrent Neural Networks for Finance

In this chapter, the TTD results from the previous chapter are extended to address the high complexity costs of RNNs. RNNs represent the de facto standard ML tool for sequence modelling, owing to their expressive power and memory. However, when dealing with large dimensional data, the corresponding exponential increase in the number of parameters imposes a computational bottleneck. The necessity to equip RNNs with the ability to deal with the curse of dimensionality, such as through the parameter compression ability inherent to tensors, has led to the development of the TT RNN. Despite achieving promising results in many applications, the full potential of the TTRNN is yet to be explored in the context of interpretable financial modelling, a notoriously challenging task characterized by multi-modal data with low signal-to-noise ratio. To address this issue, we investigate the potential of TTRNN in the task of financial forecasting of currencies. We show, through the analysis of TT-factors, that the physical meaning underlying TD, enables the TTRNN model to aid the interpretability of results, thus mitigating the notorious "black-box" issue associated with neural networks. Furthermore, simulation results highlight the regularization power of TT decomposition, demonstrating the superior performance of TTRNN over its uncompressed RNN counterpart and other tensor forecasting methods.

4.1 Introduction

RNNs [69] have proven to be among the most successful ML approaches for sequence modelling. Despite considerable success, RNNs are known to be difficult to train when considering large-dimensional inputs, due to an exponential increase in the number of parameters [70, 71]. One way to tackle this issue is through the compression properties of TD.

Tensors are a multi-linear generalization of vectors and matrices to multi-way arrays [20, 21]. Similar to their matrix counterparts, latent factors in tensors can be extracted via TD, which can be employed for compression and to obtain low-rank representations of multi-way data [21, 22]. These desirable properties, combined with the fact that tensors can be readily constructed from vectors and matrices

via tensorization [18, 19], make them particularly attractive to the data analytics communities.

Although substantial research has been carried out on the topic [72, 73, 74, 75], tensors have firstly been considered in conjunction with neural networks only in 2015 [28], where the authors employed TTD [49] to drastically compress a fully-connected neural network, while maintaining comparable performance. This was later extended to TTRNNs in [29, 76, 77] for modelling sequential data. Despite promising results, the full potential stemming from the combination of tensors and neural networks is yet to be explored, especially in the area financial forecasting.

Financial forecasting is a notoriously difficult task, characterized by multi-way data with low signal-to-noise ratio [78, 79, 80]. In the high-frequency trading domain, several deep learning and tensor based algorithms have demonstrated promising results by learning from large amounts of order-book data [81, 82, 83, 84, 85]. However, at a daily trading frequency, the amount of available data becomes too scarce compared to its dimensionality, which poses significant challenges to these learning algorithms in terms of over-fitting [86, 87].

We here show that the TTRNN offers a plausible solution to the notoriously difficult task of low-frequency financial forecasting. Indeed, the multi-way nature of financial data naturally admits a tensor representation, which is inherent to the tensorized nature of TTRNN. In addition, the super-compression property of the TTD drastically reduces the space complexity of the RNN, which can be used as a regularization tool that directly controls over-fitting, while preserving the underlying data structure and interpretability. This motivates us to employ a TTRNN to the task of financial forecasting of currencies, a task which involves multi-way financial data from a range of assets and asset classes. We also show how the inherent expressiveness of tensors enables the TTRNN to offer new perspectives on the interpretability of results, by indicating which TT-cores, and consequently which tensor modes [74], play a more prominent role in forecasting. This both mitigates the "black-box" nature of neural networks, and offers enhanced results over its uncompressed RNN counterpart and other tensor models, as illustrated in experiments.

4.2 Preliminaries

4.2.1 *Tensor-Train Fully Connected Neural Networks*

A fully connected layer in neural networks applies an element-wise activation function, $\sigma(\cdot)$, on the linear transformation to an input vector $\boldsymbol{x} \in \mathbb{R}^P$, that is

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{4.1}$$

with a weight matrix, $\boldsymbol{W} \in \mathbb{R}^{M \times P}$, and a bias vector, $\boldsymbol{b} \in \mathbb{R}^M$. If the dimensions of the typically large weight matrix can be factorized as $P = \prod_{n=1}^{N} I_n$ and $M = \prod_{n=1}^{N} J_n$, for $i_n = 1, \ldots, I_n$ and $j_n = 1, \ldots, J_n$, then the linear transformation (4.1) can be

expressed in tensor form as:

$$\boldsymbol{\mathcal{Y}}\big[j_1,\ldots,j_N\big] = \sum_{i_1,\ldots,i_N} \boldsymbol{\mathcal{W}}\big[(i_1,j_1),\ldots,(i_N,j_N)\big]\boldsymbol{\mathcal{X}}\big[i_1,i_2,\ldots,i_N\big] + \boldsymbol{\mathcal{B}}\big[j_1,\ldots,j_N\big] \tag{4.2}$$

where $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1\times\cdots\times I_N}$, $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{J_1\times\cdots\times J_N}$, $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{J_1\times\cdots\times J_N}$ are order-N tensorizations of $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{b}$ respectively, while $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{(I_1\times J_1)\times\cdots\times(I_N\times J_N)}$ is the order-$2N$ tensorization of $\boldsymbol{W}$ [29]. The so tensorized weight matrix can now be decomposed via the MPO definition of the TT-Decomposition [28] [29] (see Section 2.1 for more details), given by

$$\boldsymbol{\mathcal{W}}\big[(i_1,j_1),\ldots,(i_N,j_N)\big] = \boldsymbol{\mathcal{G}}^{(1)}\big[\,:,i_1,j_1,:\,\big]\cdots\boldsymbol{\mathcal{G}}^{(N)}\big[\,:,i_N,j_N,:\,\big] \tag{4.3}$$

4.2.2 *Tensor-Train Recurrent Neural Networks*

RNNs capture time-varying dependencies via feedback weights that process information related to previous time steps [69], that is

$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}^{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}^{xh}\boldsymbol{x}_t + \boldsymbol{b}^h) \tag{4.4}$$

where the hidden state at time t, $\boldsymbol{h}_t \in \mathbb{R}^M$, is generated as a function of: (i) the previous hidden state $\boldsymbol{h}_{t-1} \in \mathbb{R}^M$, (ii) the input $\boldsymbol{x}_t \in \mathbb{R}^P$, (iii) the feedback weight matrix $\boldsymbol{W}^{hh} \in \mathbb{R}^{M\times M}$, (iv) the input weight matrix $\boldsymbol{W}^{xh} \in \mathbb{R}^{M\times P}$, and (v) a bias vector $\boldsymbol{b}^h \in \mathbb{R}^M$; $\sigma(\cdot)$ is an element-wise activation function.

Similar to the application of TT to fully-connected neural networks, the folding of $\boldsymbol{h}_t$, $\boldsymbol{x}_t$, and $\boldsymbol{b}$ into respective order-N tensors, $\boldsymbol{\mathcal{H}}_t \in \mathbb{R}^{J_1\times J_2\times\cdots\times J_N}$, $\boldsymbol{\mathcal{X}}_t \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$, and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{J_1\times J_2\times\cdots\times J_N}$, allows us to express $\boldsymbol{W}^{xh}$ in the TT-format as in (4.3). This results in the TTRNN model [29], which extends the notion of tensorized neural networks to sequence modelling.

Remark 6 The TTRNN compresses the parameter complexity for $\boldsymbol{W}^{xh}$ from an exponential $\prod_{n=1}^{N} I_n J_n = PM$ to a linear $\sum_{n=1}^{N} I_n J_n R_{n-1} R_n$ in the dimensions I_n and J_n, which is highly efficient for small mode sizes and TT-ranks. This improves the storage complexity, computational complexity, and training time [29].

4.3 Tensor Train Recurrent Neural Network for Finance

4.3.1 *Input Data Structure*

To provide a comprehensive and multi-modal scheme to the task of financial forecasting of the Japanese Yen currency (symbol JPYUSD), we considered 24 financial signals with multi-way structure from Bloomberg, as shown in Table 4.1. The considered financial signals span four different asset classes: equities, currencies, commodities, and fixed income, with each class comprising of six financial components. The input feature vector at time instant, t, for i-th financial signal, $\boldsymbol{x}_t^{(i)} \in \mathbb{R}^{20}$, consists of 20 features:

1. Daily log-differences, $\boldsymbol{r}_t = \log(\boldsymbol{p}_t) - \log(\boldsymbol{p}_{t-1})$, where $\boldsymbol{p}_t$ is the daily closing value (**1 feature**);

2. Rolling statistics of log-differences, such as the average, standard deviation, skewness, and kurtosis over $5, 10, 22$ time-steps, corresponding to 1 week, 2 weeks, and 1 month of trading data (**12 features**);

3. Relative price min-max $\frac{\boldsymbol{p}_t - \min_{t\in T}(\boldsymbol{p}_t)}{\max_{t\in T}(\boldsymbol{p}_t) - \min_{t\in T}(\boldsymbol{p}_t)}$, where $T = \{t - L + 1, \ldots, t-1, t\}$ and L is a window of 5, 10, and 22 time-steps (**3 features**);

4. Relative high-low-close, $\frac{\boldsymbol{p}_t - \boldsymbol{l}_t}{\boldsymbol{h}_t - \boldsymbol{l}_t}$, where $\boldsymbol{l}_t$ and $\boldsymbol{h}_t$ are the low and high price for day t (**1 feature**);

5. High-low spread, $\frac{\boldsymbol{h}_t - \boldsymbol{l}_t}{\boldsymbol{l}_t}$ (**1 feature**);

6. Units traded, i.e. the volume, at time t (**1 feature**);

7. Units of unsettled future contracts at time t (**1 feature**).

For every time-step, t, all 24 feature vectors $\boldsymbol{x}_t^{(i)} \in \mathbb{R}^{20}$ were arranged into a 3-rd order tensor, $\boldsymbol{\mathcal{Z}}_t \in \mathbb{R}^{20\times 6\times 4}$ (features $\times$ components $\times$ asset classes), which was further reshaped into a 5-th order tensor, $\boldsymbol{\mathcal{X}}_t \in \mathbb{R}^{2\times 2\times 5\times 6\times 4}$, such that the first three modes correspond to the 20 features, while the last two correspond to the asset class components and asset classes. An intuitive illustration portraying the multi-modal nature of financial data in tensor form is presented in Figure 4.1.

Remark 7 The choice of a 5-th order tensor, $\boldsymbol{\mathcal{X}}_t$, instead of the original 3-rd order tensor, $\boldsymbol{\mathcal{Z}}_t$, improves the compression due to smaller mode sizes. This comes without loss in physical interpretability, as the modes corresponding to features, components, and asset classes are not mixed.

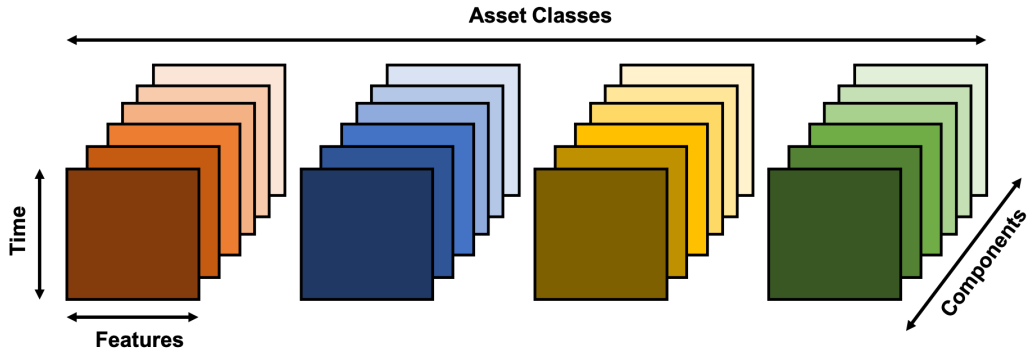


Figure 4.1: Financial data represented as a multi-modal tensor.

Table 4.1: Multi-way financial data selected from Bloomberg

Asset Class	Symbol	Description
Equities (Index)	SPX	US stock index
Equities (Index)	MXCA	Canadian stock index
Equities (Index)	UKX	UK stock index
Equities (Index)	FTSEMIB	Italian stock index
Equities (Index)	SHSZ300	Chinese stock index
Equities (Index)	NKY	Japanese stock index
Currencies (FX Pair)	CHFUSD	Swiss Frank
Currencies (FX Pair)	CADUSD	Canadian Dollar
Currencies (FX Pair)	GBPUSD	British Pound
Currencies (FX Pair)	EURUSD	Euro
Currencies (FX Pair)	CNYUSD	Chinese Yuan
Currencies (FX Pair)	JPYUSD	Japanese Yen
Commodities (Futures)	GC1	Gold
Commodities (Futures)	HG1	Copper
Commodities (Futures)	CL1	Crude Oil
Commodities (Futures)	NG1	Natural Gas
Commodities (Futures)	S1	Soybeans
Commodities (Futures)	C1	Corn
Fixed Income (Index)	USGG10YR	US 10 Yrs Yield
Fixed Income (Index)	GCAN10YR	Canadian 10 Yrs Yield
Fixed Income (Index)	GUKG10	UK 10 Yrs Yield
Fixed Income (Index)	GBTPGR10	Italian 10 Yrs Yield
Fixed Income (Index)	GCNY10YR	Chinese 10 Yrs Yield
Fixed Income (Index)	GJGB10	Japanese 10 Yrs Yield

4.3.2 *Model Training Settings*

We considered the financial data in Table 4.1 in the period from May 2006 to May 2019. The first 90% of data was used for training, and the remaining 10% for testing. Each feature was centred about its mean and normalized by its standard deviation. We employed a neural network consisting of one recurrent layer with 1024 units and tanh activation, and one fully connected layer with 3 units and softmax activation. The output labels $\{+1, 0, -1\}$ were assigned to the Japanese Yen time-series, according to whether the price at time step $(t + 1)$ increased $(+1)$ or decreased (-1) with respect to time step t; the value 0 was assigned if $|r_t| \leq 10^{-4}$ to avoid unnecessary transaction costs in trading small market movements. Each input sample consisted of 10 time-steps of $\boldsymbol{\mathcal{X}}_t$.

We employed the TTRNN model in the recurrent layer to compress the weight matrix $\boldsymbol{W}^{xh} \in \mathbb{R}^{M \times P}$ by reshaping and expressing it in the TT-format, in accordance to the dimensions of the input tensors $\boldsymbol{\mathcal{X}}_t$, such that $P = 2 \times 2 \times 5 \times 6 \times 4$ and $M = 4 \times 4 \times 4 \times 4 \times 4$. An illustration of the interaction between $\boldsymbol{\mathcal{X}}_t$ and the tensorized $\boldsymbol{W}^{xh}$ is shown in Fig. 4.2 in TN form.

The so established TTRNN model was trained using stochastic gradient descent with a learning rate of 10^{-5} over 20 epochs, with a batch size of 66, and using the

categorical cross-entropy loss function. The TT-ranks (R_1, R_2, R_3, R_4) for the TTRNN model were empirically found to be optimal for rank $R_i = 6$, $i = 1, \ldots, 4$. It was also found that higher values of R_i would result in over-fitting, and lower values in under-fitting. In turn, this suggests that the TT-ranks may be used for regularization in addition to compression.

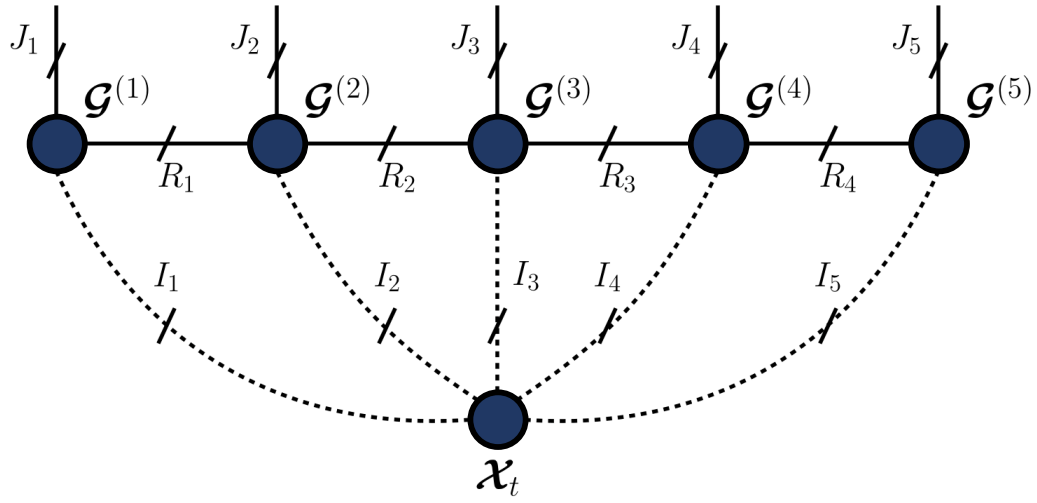


Figure 4.2: Modal connections between every mode of the input data tensor, $\boldsymbol{\mathcal{X}}_t$, and the TT-cores, $\boldsymbol{\mathcal{G}}_n$, of the tensorized weight matrix $\boldsymbol{W}^{xh}$ in TT-format.

Remark 8 Fig. 4.2 indicates that by keeping track of the variation of each core during training allows the TTRNN model to: (i) reveal information about the relative importance of each mode, and (ii) achieve network compression and regularization [74].

4.3.3 *Experimental Results*

For the given task, we compared the performance of the proposed TTRNN against: (i) its uncompressed RNN counterpart, (ii) a TT fully-connected neural network (TTNN) [28], and (iii) a Support Tensor Machine (STM) [88].

Fig. 4.3 shows the simulated cumulative profits generated by trading according to model predictions over the test period. The trades were updated daily and sized proportionally to the probability estimates of the model. A buy-and-hold strategy of the underlying asset JPYUSD is also shown in the figure.

To evaluate the performance of each model, we considered: (i) the annualized Sharpe ratio of the profits generated in the test-period (defined as $\sqrt{252}\frac{u_r}{\sigma_r}$, where u_r is the average daily returns and σ_r is the standard deviation of daily returns), (ii) total returns generated, and (iii) the directional accuracy of the predictions. The results are summarized in Table 4.2.

By unifying the regularization property of tensors and the sequence modelling ability of RNNs, the proposed TTRNN out-performed its uncompressed RNN counterpart and other general purpose tensor models such as TTNN and STM, obtaining the best experimental results across all metrics.

Remark 9 Due to the low signal-to-noise ratio of financial data, a classification accuracy of above 50% is considered significant for financial applications.

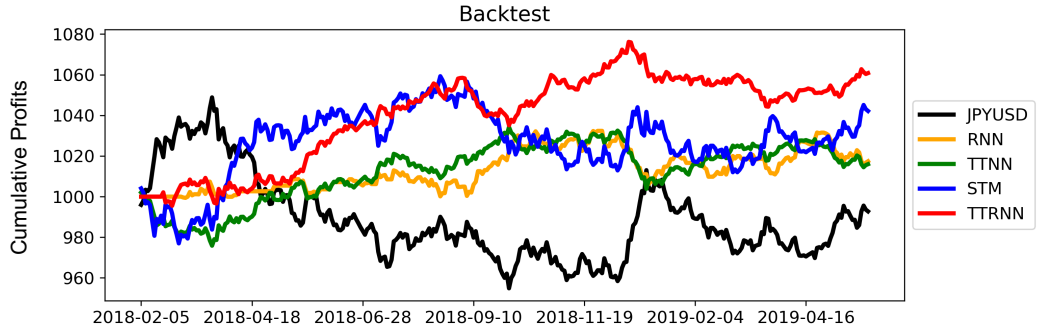


Figure 4.3: Cumulative profits generated by the models.

	Sharpe	Total Return	Accuracy
JPYUSD	-0.014	-0.73%	N.A.
RNN	0.491	1.74%	36.05%
TTNN	0.372	1.60%	51.06%
STM	0.520	4.21%	51.67%
TTRNN	**1.548**	**6.08%**	**52.04%**

Table 4.2: Performance comparison of the considered models.

4.3.4 *Interpreting Relative Modal Importance*

In addition to the compression and regularization power of the TT decomposition, the multi-modal structure inherent to the TT allowed us to interpret the relative modal importance of the data, by indicating which data modality experienced largest changes during training. This is illustrated in Fig. 4.4, which shows the normalized TT-core change over the training epochs, defined as

$$\frac{||\Delta \mathcal{G}_n^e||_F^2}{I_n J_n R_{n-1} R_n} \tag{4.5}$$

where $\Delta \mathcal{G}_n^e = \mathcal{G}_n^e - \mathcal{G}_n^{e-1}$ and $\mathcal{G}_n^e$ is the value of $\mathcal{G}_n$ at epoch $e = 2, \ldots, 20$.

The cores $\mathcal{G}_1$ and $\mathcal{G}_5$ exhibited greatest changes, suggesting that the hand-crafted features contained in mode 1 (see Section 4.3.1), and the asset classes contained in

mode 5 (see Table 4.1), account for most of the predictive power of TTRNN for our problem. Also, the greater change in $\mathcal{G}_5$ over $\mathcal{G}_4$ suggests greater importance of inter-class dependencies (mode-5) over intra-class financial signals (mode-4). This insight is only possible with the weights represented in the TT format, which associates a TT core to each of the data modality separately, hence bringing the benefits of interpretability in addition to the compression and regularization properties of TTD.

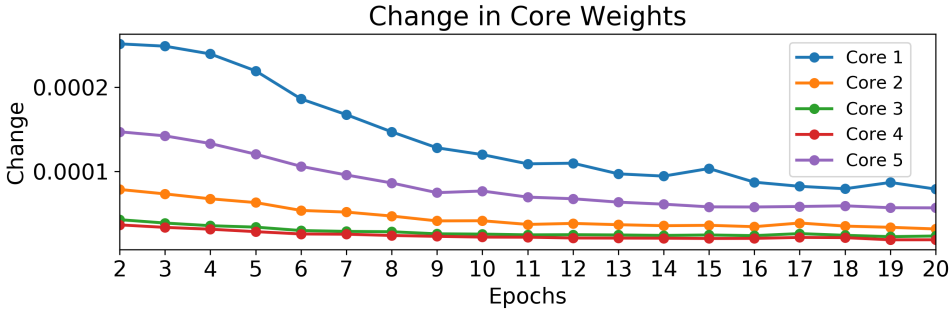


Figure 4.4: Normalized change of TT-core weights in TTRNN.

4.4 Summary

Traditional currency forecasting models are typically based on intra-class variables only, despite the wide availability of other strongly or weakly related asset classes. To address this issue, we investigated the ability of TTRNNs to perform financial forecasting of currencies from multi-class data. This has been achieved based on multi-way financial data from four asset classes: equities, currencies, commodities, and fixed income. The model has been trained on tensorized data from May 2006 to February 2018, and tested on the period from February 2018 to May 2019. Simulation results have verified the desirable compression and regularization properties of the TTRNN, which outperformed its uncompressed RNN counterpart and other tensor based forecasting methods. Moreover, through the gradient inspection of individual TT cores, TTRNN has been demonstrated to allow for physical interpretability of relative modal importance in a financial context, thus mitigating the well-known black-box issue inherent to neural networks.

Part II

Graph Learning Methods

Chapter 5

Graph Theory for Metro Traffic Modelling

Part II of this thesis commences by introducing various graph theory concepts, accompanied by real-life applications that offer intuitive insights. In particular, a unifying graph theoretic framework for the modelling of metro transportation networks is proposed. This is achieved by first introducing a basic graph framework for the modelling of the London underground system from a diffusion law point of view. This forms a basis for the analysis of both station importance and their vulnerability, whereby the concept of graph vertex centrality plays a key role. We next explore k-edge augmentation of a graph topology, and illustrate its usefulness both for improving the network robustness and as a planning tool. Upon establishing the graph theoretic attributes of the underlying graph topology, we proceed to introduce models for processing data on such a metro graph. Commuter movement is shown to obey the Fick's law of diffusion, where the graph Laplacian provides an analytical model for the diffusion process of commuter population dynamics. Finally, we also explore the application of modern deep learning models, such as GNNs and hyper-graph GNNs, as general purpose models for the modelling and forecasting of underground data, especially in the context of the morning and evening rush hours. Comprehensive simulations including the passenger in- and out-flows during the morning rush hour in London demonstrates the advantages of the graph models in metro planning and traffic management, a formal mathematical approach with wide economic implications.

5.1 Introduction

The rapid development of many world's economies has been followed by an increasing proportion of population moving to cities [89, 90]. Through a "domino-effect" of such a huge societal change, world's urban traffic congestion has become, and is likely to remain, an overwhelming issue. In addition, underground traffic networks frequently experience signal failures and train derailments - not to mention emergency measures due to accidents - while closure of one station may impact the service across the entire network. The economic costs of such transport delays to central London business are estimated to be $£1.2$ billion per year. Appropriate and physically meaningful tools

to understand, quantify, and plan for the resilience of transportation networks to disruptions are therefore a pre-requisite for the planning and daily running of public transport, a subject of this work.

Graph representations for traffic planning have a long history [91, 92, 93, 94, 95]; as far back as in 1926, a map-maker named Fred Stingemore produced a map of the London underground by regularising the spacing between stations, while allowing himself some artistic freedom with the routes of the various lines. Stingemore's work was further abstracted by Harry Beck in 1933 to the graph form we have today. In this chapter, we move beyond the graphs themselves and employ the concepts of Data on Graph [35, 34, 36], to address some burning issues in underground public transport, such as "sensitivity" of the network to station closures. We further demonstrate how graph theory can be used to identify those stations in the London underground network which have the greatest influence on the functionality of the traffic, and proceed, in an innovative way, to assess the impact of a station closure on service levels across the city. Such underground network vulnerability analysis is shown to provide the opportunity to analyse, optimize and enhance the connectivity of the traffic networks in a mathematically tractable and physically meaningful way. We also explore the benefits of modelling the London underground graph via modern deep learning approaches, through leveraging models such as GNNs and hyper-graph GNNs to perform semi-supervised learning on graphs.

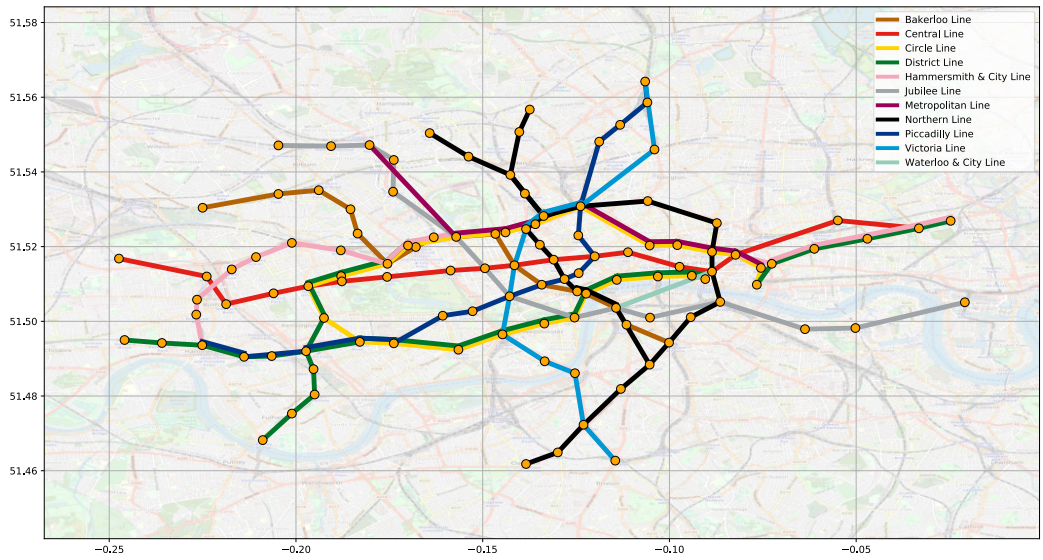


Figure 5.1: Graph representation of the London underground network in Zones 1 and 2. The circles denote the vertices (stations) and the lines between the circles designate the underground lines.

5.2 Traffic Centrality and Vulnerability

We consider the underground network as an undirected N-vertex graph, denoted by $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, with $\mathcal{V}$ as the set of N vertices (stations) and $\mathcal{E}$ the set of edges (underground lines) connecting the vertices (stations). The connectivity of the network is designated by the (undirected) adjacency matrix, $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, where $\boldsymbol{A}_{ij} = 1$ if a connection between the i-th and j-th station exists, and 0 otherwise [35]. Figure 5.1 shows the proposed graph model of the London underground network, with each vertex representing a station, and each edge designating the the connection between two adjacent stations through one or more underground lines. For visualization purposes, we show the graph representation for the central zones 1-2, out of the 6 zones of the London underground network. Notice that standard data analytics typically assume regular grids in time and in space, and are thus ill-equipped to deal with this class of problems, which reside on irregular domains that may exhibit multiple paths between adjacent vertices (stations). This inadequacy becomes even more prominent during the "anomaly" times, such as the morning an evening rush hours.

The graph representation of the traffic network allows us to explore the usefulness of the betweenness centrality metric as a measure of the vulnerability of a certain station to traffic disruption. This metric is quite natural in the context of transportation planning, as the betweenness centrality measures the extent to which a given vertex lies in between pairs or groups of other vertices, and is given by $b(n) = \sum_{k,m \in \mathcal{V}} \frac{\sigma(k,m|n)}{\sigma(k,m)}$ where $\sigma(k,m)$ denotes the number of shortest paths between vertices k and m, and $\sigma(k,m|n)$ the number of those paths passing through a vertex n [96]. In other words, betweenness centrality reflects the extent to which a vertex (station) is an intermediate in the communication over the graph (underground network). Figure 5.2 depicts the values of betweenness centrality of stations in the London underground network in Zones 1 and 2. Observe that, as expected, the stations at the centre of the city exhibit the largest betweenness centrality, while the stations in the outer residential area have the smallest betweenness centrality. Physically, this means that disruptions to lines passing through stations with high betweenness centrality would cause less impact on the overall network, as there are alternative ways of accessing the same station via other underground lines. Conversely, if connections to low centrality stations were to be disrupted, it would cut-off both that station and those further out from accessing the rest of the network.

Table 5.1: Stations with largest betweenness centrality.

Station	Entries	Exits	Net out-flow
Bank	$17,577$	$69,972$	$52,395$
Canary Wharf	$8,850$	$56,256$	$47,406$
Oxford Circus	$3,005$	$44,891$	$41,886$

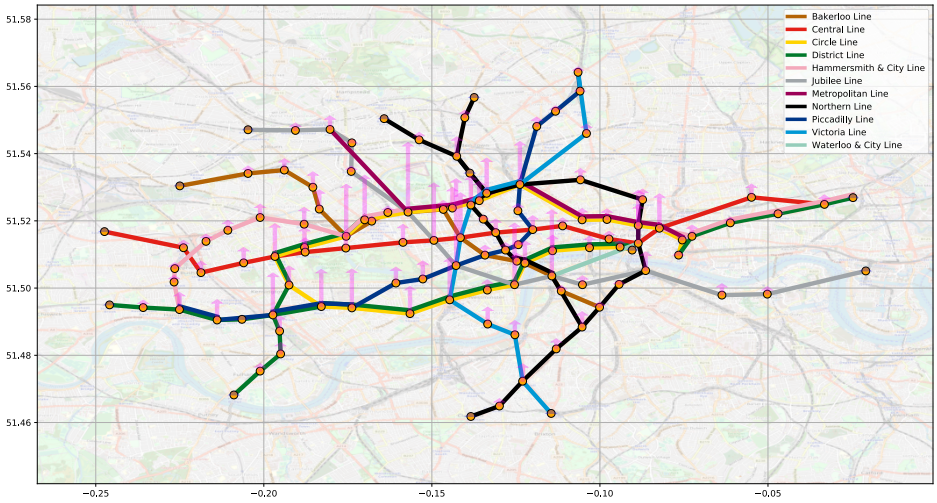


Figure 5.2: Betweenness centrality, designated by magenta-coloured bars, of the London underground network in Zones 1 and 2. The largest betweenness centrality is observed for at stations with multiple metro lines parsing through.

5.3 K-Edge Augmentation for Robust Traffic Networks

Given that the partial closure of a line between any two metro stations will severely impact the entire metro system [97], we set out to analyse the vulnerability of the metro system in terms of the level of connectivity. To this end, we introduce a measure of the underground network robustness in terms of k-edge connectivity. Formally, a graph is said to be k-edge connected if it cannot be regrouped into distinct sub-graphs unless k or more edges are removed. For the London underground network in Zones 1 and 2 in Figure 5.1, the graph topology forms a 1-edge connected graph. This implies that a removal of even one connection (between two adjacent stations) suffices to disconnect some stations from the rest of the network. It then comes as no surprise that the stations exhibiting lowest betweeness centrality (see Figure 5.2) are most vulnerable to the closure of other stations.

Our solution for improving the robustness of the overall network is based on k-edge augmentation, a search problem for determining the minimum set of additional edges, $\mathcal{A}$, such that $\mathcal{A} \cap \mathcal{E} = \varnothing$ and the resulting graph $\mathcal{G}_k = (\mathcal{V}, \mathcal{E} \cup \mathcal{A})$ remains k-edge connected [98]. Figure 5.3 shows an example of k-edge augmentation for the London underground network for $k = 2$. Although the solution presented in Figure 5.3 is optimal in terms of the number of new connections required, $|\mathcal{A}|$, it is highly unrealistic. To this end, assuming that the cost of building a new connection is proportional to the geographic distance between two stations, we can constrain the search space of the k-edge augmentation problem so as to include only new connections which are quite short, that is $d(v_1, v_2) < \alpha$ for all $(v_1, v_2) \in \mathcal{A}$, where $d(\cdot)$ denotes the geographic distance between any two stations and α is a user chosen threshold. Figure 5.4 shows such a physically more realistic augmented network.

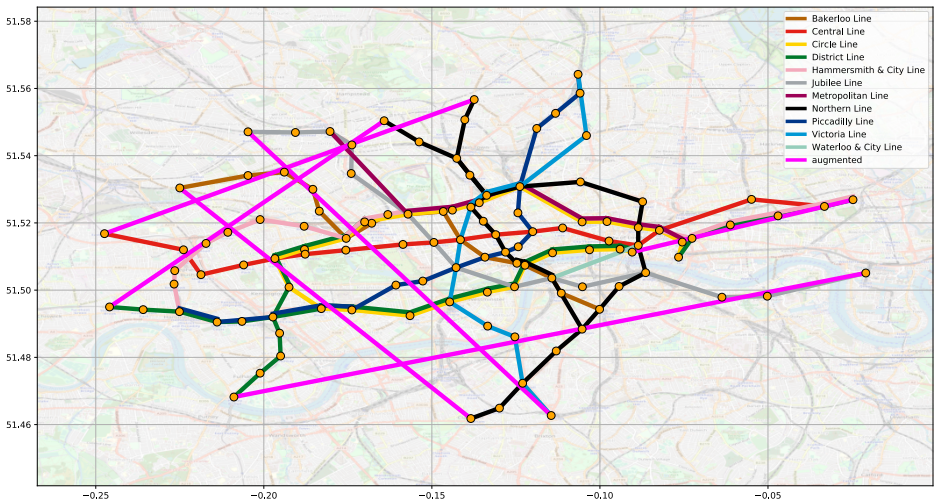


Figure 5.3: Graph of the London underground network in Zones 1 and 2, after performing a naive k-edge augmentation, for k=2, to improve the robustness of the network. The new lines are designated in magenta, while k=2 means that no station can be disconnected from the rest of the network if only one underground line passing through that station is not operational.

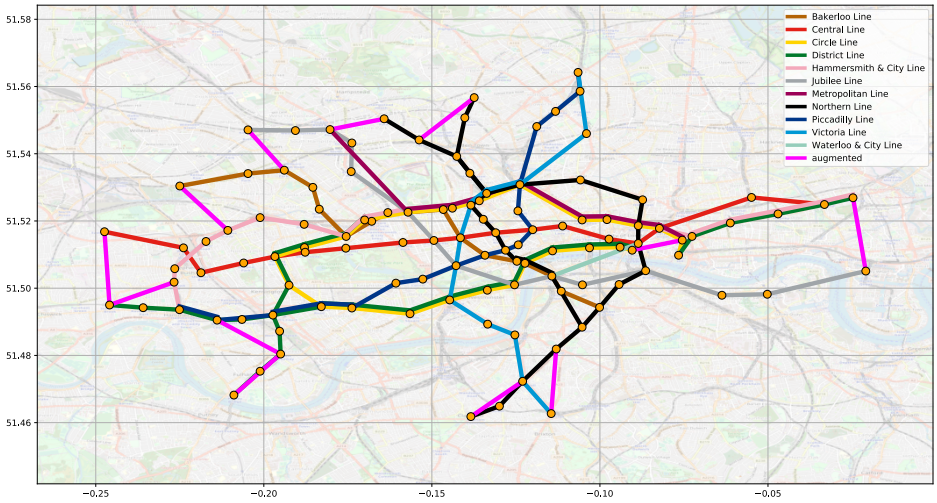


Figure 5.4: Graph of the London underground network in Zones 1 and 2 after performing geographically constrained k-edge augmentation, for k=2, to improve the robustness of the network. Observe that the new segments, designated in magenta, are of similar length as the already existing segments.

5.4 Modelling Net Passenger Flow and Population

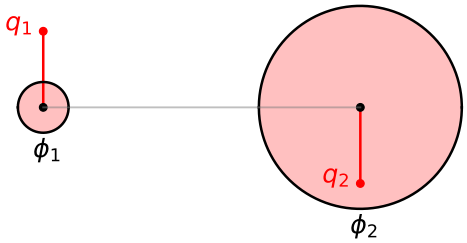


Figure 5.5: A graph representation of the London underground network through Fick's law of diffusion. Consider a simplified path graph with two stations surrounded by the respective populations, ϕ_1 and ϕ_2 (proportional to the circle area), which exhibit the corresponding net fluxes, q_1 and q_2. Stations surrounded by large population (residential areas) exhibit a net in-flow of passengers, while stations surrounded by low population (business districts) experience net out-flow of passengers. The overall net flow (in-flow and out-flow) of passengers across the entire network sums up to zero.

We shall now demonstrate that the graph theoretic approach to transportation modelling allows us to answer even broader societal questions, such as to infer the resident population surrounding each station solely based on the net passenger flow during the morning rush hour.

To derive the corresponding graph model, we employ the Fick's law of diffusion which relates the diffusive flux to the concentration of a given vector field, under the assumption of a steady state. According to Fick's law, the flux flows from regions of high concentration (population) to regions of low concentration (population), with a magnitude proportional to the concentration gradient. Mathematically, the Fick law is given by

$$\boldsymbol{q} = -k\nabla\boldsymbol{\phi} \tag{5.1}$$

where $\boldsymbol{q}$ is the flux which measures the amount of substance per unit area per unit time (mol m^{-2} s^{-1}), k is the coefficient of diffusivity measured in area per unit time (m^2 s^{-1}), while $\boldsymbol{\phi}$ represents the concentration (mol m^{-3}).

This framework allows us to model the passenger flow in the London underground network as a diffusion process, whereby during the morning rush hour the population mainly flows from population-dense residential areas to sparsely populated business districts. Based on the general Fick's law in (5.1), the variables in our proposed metro traffic model are: (i) $\boldsymbol{q} \in \mathbb{R}^N$, the net passenger flow vector, with the i-th entry as the net passenger flow at the i-th station during the morning rush hour, given by: $\boldsymbol{q}_i = \ln(\boldsymbol{p}_i^{(o)}) - \ln(\boldsymbol{p}_i^{(i)})$, where $\boldsymbol{p}_i^{(o)}$ and $\boldsymbol{p}_i^{(i)}$ denote respectively the number of passengers exiting and entering the i-th station, with its dimension equal to "passengers per station per unit time"; (ii) $k = 1$, the coefficient of diffusivity, with its dimension equal to "stations per unit time"; (iii) $\boldsymbol{\phi} \in \mathbb{R}^N$, which represents the resident population in the area surrounding the station.

The proposed graph-theoretic model is physically meaningful and suggests that, in the morning, the net passenger flow at the i-th station, $\boldsymbol{q}_i$, is proportional to the population difference between the areas surrounding a station i and the adjacent stations j, that is

$$\begin{aligned}
\boldsymbol{q}_i &= -k \sum_j \boldsymbol{A}_{ij}(\boldsymbol{\phi}_i - \boldsymbol{\phi}_j) \\
&= -k \left(\boldsymbol{\phi}_i \sum_j \boldsymbol{A}_{ij} - \sum_j \boldsymbol{A}_{ij}\boldsymbol{\phi}_j \right) \\
&= -k \left(\boldsymbol{\phi}_i \boldsymbol{D}_{ii} - \sum_j \boldsymbol{A}_{ij}\boldsymbol{\phi}_j \right) \\
&= -k \sum_j \left(\boldsymbol{\delta}_{ij}\boldsymbol{D}_{ii} - \boldsymbol{A}_{ij} \right) \boldsymbol{\phi}_j \\
&= -k \sum_j \boldsymbol{L}_{ij}\boldsymbol{\phi}_j
\end{aligned} \tag{5.2}$$

The model in (5.1) also allows us to jointly consider N stations in matrix form

$$\boldsymbol{q} = -k\boldsymbol{L}\boldsymbol{\phi} \tag{5.3}$$

where $\boldsymbol{L} = (\boldsymbol{D} - \boldsymbol{A}) \in \mathbb{R}^{N \times N}$ is the Laplacian matrix of the graph in Figure 5.1 [34]. Also, the commuter dynamics model based on Fick's law has an intimate connection with the GSOs [35, 34]. For enhanced physical intuition, Figure 5.5 shows a signal within this diffusion model on a 2-vertex path graph obeying the Fick law.

The data for the average daily net flow of passengers during the morning rush hour at each station in 2016 was obtained from Transport for London (TFL)[1], and is visualised as a signal on the London underground graph model in Figure 5.6.

Table 5.2 shows the daily average net flow of passengers per Zone of the London underground network. As expected, Zone 1 (central London) is the only Zone to exhibit a net out-flow of passengers, while Zone 2 (more residential) shows a net in-flow of passengers. In an ideal scenario, the total net out-flow across all Zones should sum up to 0, however, the residual net out-flow is attributed to passengers entering the underground network through other transport services not considered in our model, such as rail services and buses.

Table 5.2: Daily average passenger flows during the morning rush hour.

Zone	Entries	Exits	Net out-flow
1	$455,704$	$844,123$	$388,419$
2	$343,145$	$264,732$	$-78,413$

[1]Available at https://tfl.gov.uk

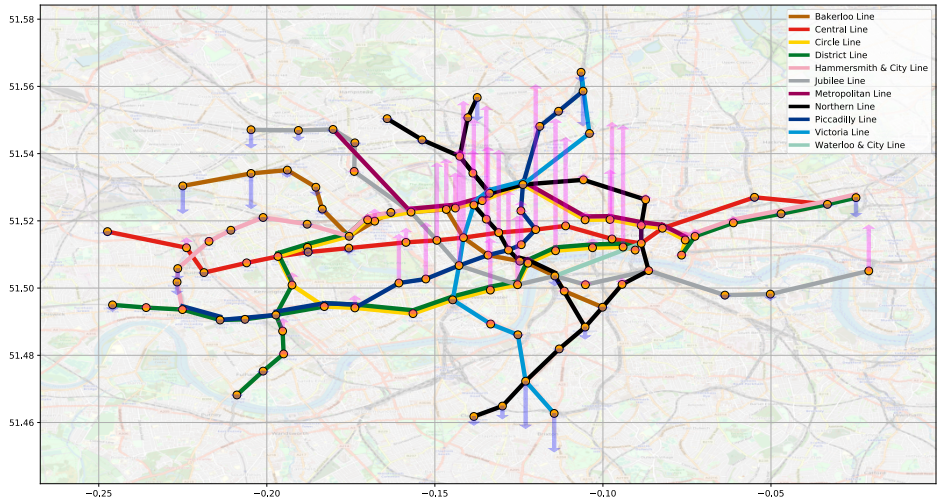


Figure 5.6: Net passenger out-flow during the morning rush hour within the Zones 1 and 2 of the London underground network (for clarity, data for only several stations in residential areas are shown). The magenta bars denote a net out-flow of passengers while the blue bars designate a net in-flow of passengers. Stations located within business districts exhibit the greatest net out-flow of passengers, while stations located in residential areas, toward the outskirts, exhibit the largest net in-flow of passengers.

Table 5.3 shows the average net flow of passengers for the top 5 stations with the greatest net in-flow and out-flow. Observe that the stations which exhibits largest net out-flow of passengers are located within the financial (Bank, Canary Wharf, Green Park) and commercial (Oxford Circus, Holborn) districts. In contrast, the stations exhibiting the greatest net in-flow of passengers are located in residential areas.

Table 5.3: Stations with largest net passenger out-flow and in-flow.

Station	**Entries**	**Exits**	**Net out-flow**
Bank	$17,577$	$69,972$	$52,395$
Canary Wharf	$8,850$	$56,256$	$47,406$
Oxford Circus	$3,005$	$44,891$	$41,886$
Finsbury Park	$20,773$	$8,070$	$-12,703$
Canada Water	$31,815$	$14,862$	$-16,953$
Brixton	$24,750$	$4,369$	$-20,381$

To obtain an estimate of the resident population surrounding each station, recall that the vector of populations, $\boldsymbol{\phi} \in \mathbb{R}^N$, is subject to (5.3), that is

$$\hat{\boldsymbol{\phi}} = -\frac{1}{k}\boldsymbol{L}^{+}\boldsymbol{q} \tag{5.4}$$

where the symbol $(\cdot)^{+}$ denotes the matrix pseudo-inverse operator. It is important to

mention that the population vector can only be estimated up to a constant, hence the vector $\hat{\phi}$ actually quantifies the relative population between stations, whereby the station with the lowest estimated surrounding population takes the value of 0. The so estimated resident population, based on the morning net passenger flow, is displayed as a signal on a graph in Figure 5.7. Observe that the so obtained estimates are reasonable, since most of the resident population in London is concentrated toward the more remote areas of Zone 2, while business districts at the centre of Zone 1 are sparsely populated in the evening.

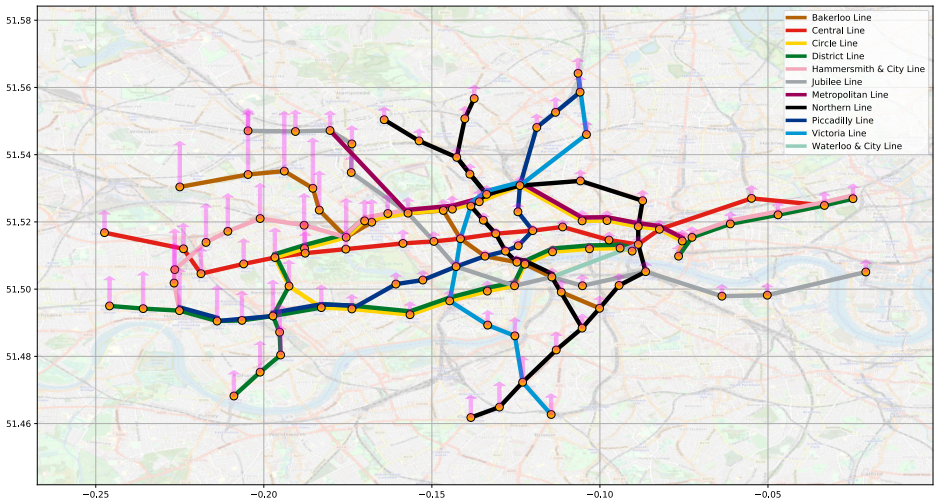


Figure 5.7: Population density implied by our graph model in (5.4), obtained from the net passenger out-flow during the morning rush hour within Zones 1 and 2. As expected, business districts exhibit the lowest population density, while residential areas (Zone 2) exhibit the highest commuter population density.

Validation of Population Estimates: The estimated population, $\hat{\phi}$, cannot be validated against ground-truth data directly, since the considered dataset does not contain population data per station. However, the net out-flow of passengers per station in the evening rush hours can be used as an indicator of population, ϕ. This is because most people head home directly after work. In our experiments, we find that $\hat{\phi}$ and ϕ are 50% correlated.

Synthetic Data Simulation: The proposed diffusion framework is further validated using synthetic data. This is achieved by sampling the ground-truth population data, ϕ, at every station using a Gaussian distribution with zero-mean and unit standard deviation. The population data is then diffused along the network with additive Gaussian noise. This will result in a graph signal of varying magnitude for each station as well as a noise of varying magnitude for each station. The implied population, $\hat{\phi}$, is then estimated from noisy traffic flow, $\boldsymbol{q}$, using (5.4). The experiment results are summarized in Table 5.4 for varying levels of Gaussian noise standard deviation. Note that the noise power is related to the square of standard deviation.

Table 5.4: Estimation MSE of diffusion implied population using synthetic data.

Noise Standard Deviation	0.1	0.2	0.3	0.4	0.5
MSE	0.0170	0.0406	0.0998	0.1953	0.2422

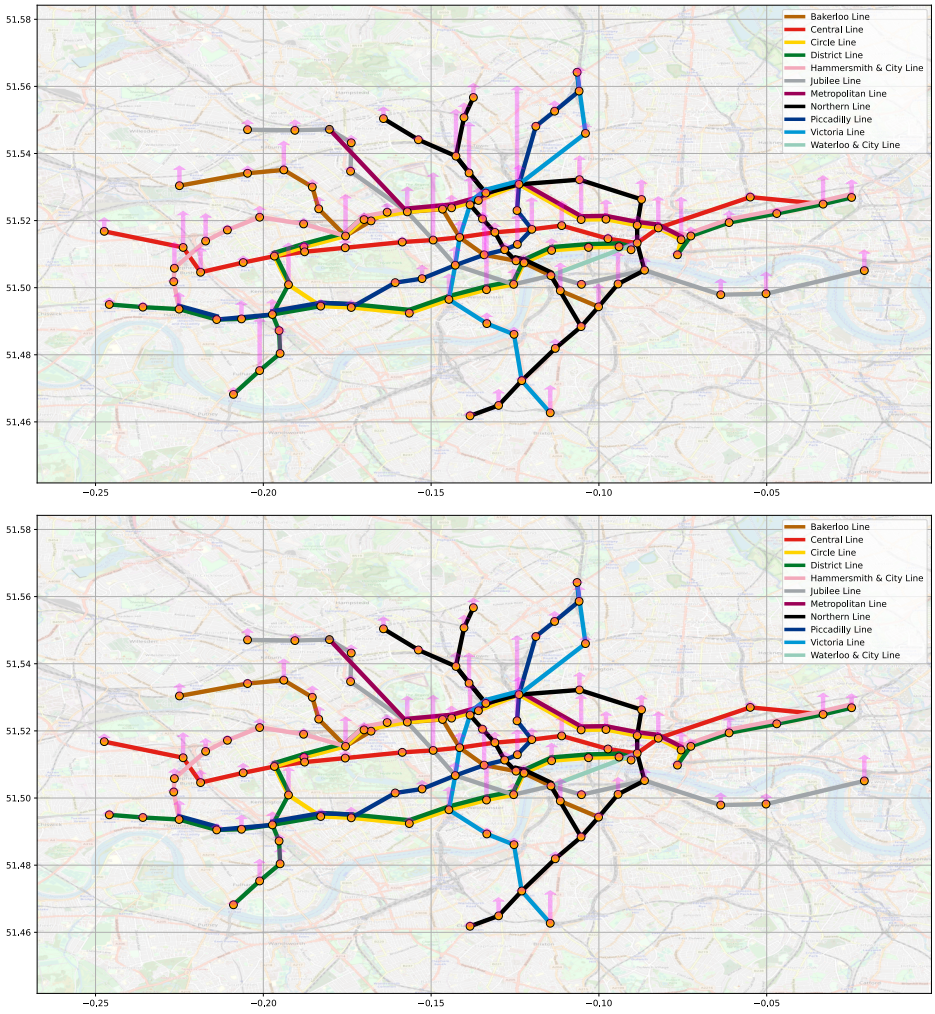


Figure 5.8: Ground-truth population (top) vs diffusion implied population (bottom) using synthetic data corrupted by Gaussian noise with 0.5 standard deviation.

5.5 A Hypergraph Model of Metro Network

The analysis of the London metro system carried out so far has been based on the classical graph adjacency matrix, $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ (and the corresponding graph Laplacian matrix $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$), where $\boldsymbol{A}_{ij} = 1$ if a connection exists between the i-th and j-th station, and 0 otherwise. However, despite its ability to capture the underlying network topology, the classical graph adjacency matrix cannot account for higher order connectivity information, such as multiple connections between the adjacent stations. To this end, we here explore the hypergraph representation of the underground network, which naturally accounts for higher order connectivity in a graph.

A hypergraph [99], $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, is a generalization of the classical graph, and is characterized by a vertex set, $\mathcal{V}$, and a hyperedge set, $\mathcal{E}$, whereby each hyperedge can contain any number of vertices. To compactly represent such a hypergraph, we shall represent $\mathcal{H}$ with the "vertex-edge" incidence matrix, $\boldsymbol{M} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{E}|}$, such that $\boldsymbol{M}_{v,e} = 1$ if the e-th edge contains the v-th vertex and 0 otherwise. Notice the difference from the "vertex-vertex" adjacency matrix, $\boldsymbol{A}$. The degree of a vertex in a hypergraph is defined as $d(v) = \sum_{e \in \mathcal{E}} \boldsymbol{M}_{v,e}$, while the degree of an edge is defined as $d(e) = \sum_{v \in \mathcal{V}} \boldsymbol{M}_{v,e}$.

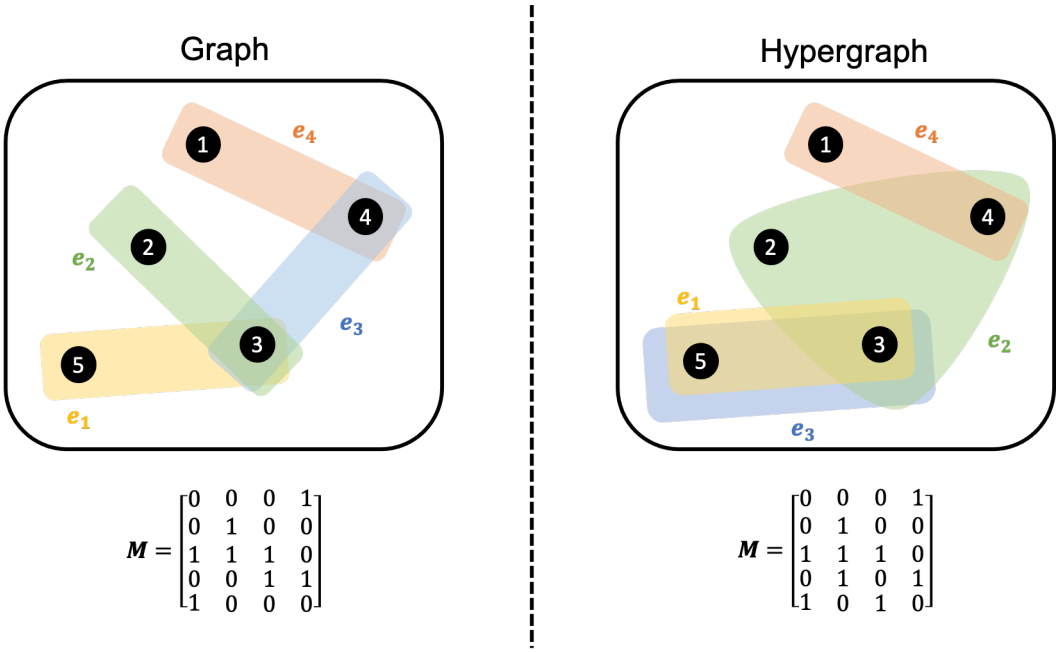


Figure 5.9: A graph (left) is a special instance of a hypergraph (right), whereby each hyperedge contains exactly two nodes. A hypergraph, which can be described by its incidence matrix, $\boldsymbol{M}$, captures high-order connections among vertices, as there is no restriction on the number of nodes per hyperedge, or the number of edges or hyperedges per vertex. For example, the hyperedge e_2 connects nodes 2, 3, and 4, as also evidenced in the corresponding incidence matrix $\boldsymbol{M}$.

The hypergraph matrix of a normalized undirected hypergraph is then

$$\boldsymbol{\Theta} = \boldsymbol{D}^{(v)^{-\frac{1}{2}}} \boldsymbol{M} \boldsymbol{D}^{e^{-1}} \boldsymbol{M}^T \boldsymbol{D}^{(v)^{-\frac{1}{2}}} \tag{5.5}$$

where $\boldsymbol{D}^{(v)} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is the vertex degree diagonal matrix composed of the individual vertex degrees, $d(v)$, for all vertices $v \in \mathcal{V}$, and $\boldsymbol{D}^{(e)} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{E}|}$ is the edge degree diagonal matrix composed of edge degrees, $d(e)$, for all $e \in \mathcal{E}$. The normalised hypergraph Laplacian matrix is defined as $\boldsymbol{\Delta} = \boldsymbol{I} - \boldsymbol{\Theta}$ [99].

To construct the hypergraph representation for the metro network, we shall simply append all multi-modal connections of the system (i.e. connections across different metro lines) as additional edges in the incidence matrix.

To validate the expressive power of the hypergraph modelling, we employ a semi-supervised regression experiment, where we estimate the net passenger out-flow during the evening rush hour for each of the underground stations, given the morning passenger flow data and the network connectivity structure. More specifically, given an input matrix, $\boldsymbol{X} \in \mathbb{R}^{N\times F}$, with F as passenger flow features for each of the N stations, we build a neural network model that estimates the vector, $\boldsymbol{y} \in \mathbb{R}^N$, of which each entry, y_i, corresponds to the evening rush hour net passenger out-flow for the i-th station.

The proposed neural network model implements a first order spatial hypergraph filter given by

$$\boldsymbol{Y} = \sigma\big((\boldsymbol{I} + \alpha\boldsymbol{\Theta})\, \boldsymbol{X}\boldsymbol{W} \big) \tag{5.6}$$

where α is a learnable graph filter coefficient, $\boldsymbol{W}$ is a learnable weight matrix performing feature extraction, and σ is an activation function. In a classical GNN implementation, we simply replace the hypergraph matrix, $\boldsymbol{\Theta}$, with the standard adjacency matrix, $\boldsymbol{A}$ [36].

Table 5.5 shows the corresponding Mean Squared Error (MSE), with both GNN models out-performing the standard neural network in the prediction of the evening traffic based on the morning rush hour data. The Hypergraph GNN model exhibited the best performance, verifying its expressivity and ability to compactly capture the multi-modal connectivity information. Figure 5.10 shows the prediction results from the proposed experiment using a hypergraph model, where the estimated evening passenger flow for each of the station is plotted on the underground graph. When measuring the out-of-sample performance, the hypergraph model achieved the lowest MSE, shown in Table 5.5, hence demonstrating its ability to capture multi-way connections between stations.

Table 5.5: Semi-supervised learning results from the proposed experiment. The MSE was calculated on the training and test set data using a $70-30\%$ split.

Model	Training MSE	Testing MSE
Neural Network	0.2736	0.3171
GNN	**0.2383**	0.2750
Hypergraph GNN	0.2400	**0.2683**

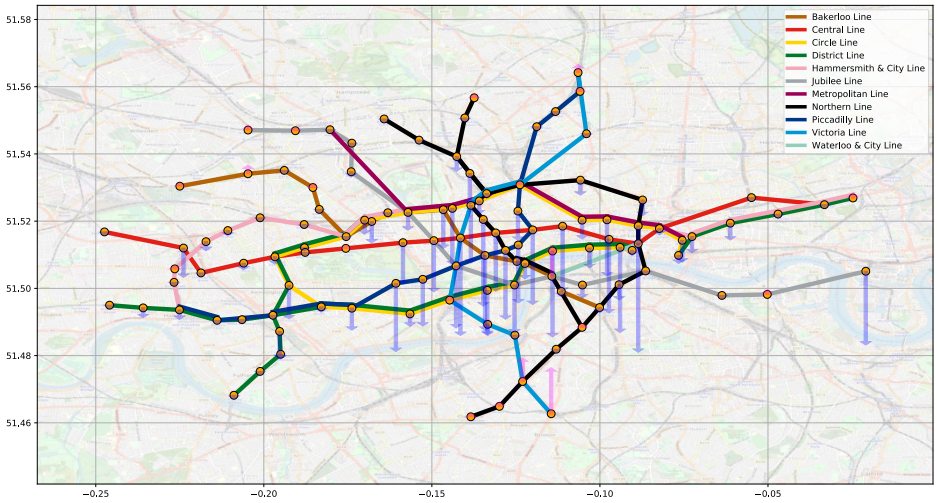


Figure 5.10: The evening net passenger-flow, estimated from the morning rush hour data, based on the hypergaph neural network model. The power of the proposed hypergraph model is reflected in the reverse net flow from the morning rush hours, as people move from central business areas back to residential zones.

5.6 Summary

We have introduced a graph-theoretic framework which makes it possible to assess the functional design and operation of public transport networks in a mathematically tractable and intuitive manner. The formulation allows not only for an assessment of the functionality, but also offers an opportunity to facilitate enhancements and assess vulnerabilities, notwithstanding the possibility of a dynamic management of the network. Our focus has been on establishing a link between the laws of diffusive fields and the parameters of urban underground traffic. In this way, we have been able to leverage on the ability of graphs to represent data on irregular domains to introduce an intuitive model for the population migration during the rush hours. Furthermore, we have employed the hyper-graph representation of the network connections, and have demonstrated that this accounts for multiple links between underground stations in a natural way, which is not achievable based on the standard graph model. Finally, we have explored the connection between graphs and modern GNNs models in this context, demonstrating the power of this approach on the prediction of the evening traffic based on the morning traffic data. It is our hope to have provided a platform to approach both traffic planning and urban transition issues from a rigorous engineering perspective using GSP techniques.

Chapter 6

Hierarchical Graph Pooling for Finance

As previously discussed in Chapter 4, the use of tensor techniques can substantially enhance neural network performance for financial modelling. Similarly, this chapter investigates the utility of GNN in the context of short-term stock price forecasting. In particular, a finance-specific graph pooling operation, referred to as *StockPool*, is introduced to efficiently coarsen the stock graph. This is achieved by employing domain knowledge to cluster stocks, depending on some task-specific characteristics (e.g. industries and sub-industries). Unlike fully end-to-end learnable graph pooling strategies (e.g. differentiable pooling and MinCUT pooling), such a deterministic pooling operator is considerably more computationally efficient and thus scalable to larger stock graphs. Experimentations on the S&P500 stock index demonstrate that the StockPool operator outperforms existing graph pooling strategies on the prediction of price movements. Finally, different graph pooling methods are utilized to create a set of highly uncorrelated GNN models; these are used to construct a graph ensemble model with an improved performance.

6.1 Introduction

In the past decade, there has been a surge in the development of GNNs for applications on irregular domains; these vary from social networks [38], traffic prediction [39, 12], through to gene inference [40]. The complex and hierarchical nature of the stock market represents therefore an ideal domain for the operation of GNNs on an irregular set of heavily structured stock graphs. However, an issue with current GNNs is that the propagation of information is limited to only between individual edges in a graph, making it difficult to learn hierarchical relationships between the nodes (e.g. stocks) in the graph. This limitation may be bypassed through graph coarsening (or pooling), which allows for a high-level representation of the stock graph at each layer of a GNN model to be obtained. This can be achieved using a subset of graph pooling operators, known as hierarchical pooling [100].

At present, there exists a variety of hierarchical pooling operators which employ different methodologies and auxiliary objectives, including Top-K pooling [101],

SAG (Self-Attention Graph) pooling [102], ASA (Adaptive Structure Aware) pooling [103], Differentiable pooling [100], MinCUT pooling [104], and the more recent Memory-based Pooling [105]. The work in [106] generalizes pooling operations under a common framework referred to as the SRC (Select, Reduce, and Connect).

The SRC framework considers the graph pooling operation as a combination of three constituent operations: *Select*, *Reduce*, and *Connect*, whereby different pooling operations employ a combination of these operations in different ways. The *Select* operation is responsible for obtaining the new supernodes of the coarsened (or pooled) graph, the *Reduce* operation obtains the new embeddings of each supernode, and finally the *Connect* operation determines the edge connections between the supernodes of the coarsened graph. A further useful categorization of graph pooling operations is regarding whether or not they are trainable. Trainable pooling operators (e.g. DiffPool [100], MinCUTPool [104], SAGPool [102], MemPool [105]) are able to learn the optimal pooling strategy in an end-to-end fashion against an objective function, while non-trainable operators (e.g. Graclus [107], NDP [108], LaPool [109]) typically use pre-defined algorithms. As suggested in [106], non-trainable methods are particularly useful when there is strong prior information regarding the most desired pooling behaviour (maintaining graph connectivity, homogeneity of node embeddings, etc.). Trainable operators, on the other hand, require fewer assumptions about the desired pooling behaviour. In the case of a stock graph, due to its unpredictable and time-varying nature, a trainable pooling operator may prove advantageous because of its ability to learn rather than assume a desired pooling behaviour [36, 110].

Note that trainable operators may also benefit from prior assumptions regarding the structure of the graph [110]. This is because the end-to-end optimization of the graph pooling strategy is non-convex [100] and thus many trainable pooling methods introduce auxiliary objective loss functions to obtain an improved convergence performance. The highly influential differential pooling operator [100], for example, uses the link prediction loss to encode the intuition that nearby nodes should be clustered together, whereas MinCUT pooling [104] introduces an auxiliary loss function to minimize the so-called mincut loss. However, while these prior assumptions have proven effective for most graph structures (e.g. social networks, road traffic networks, document citation networks), they may not perform particularly well when applied to the stock graph problem. This is due to the inherently non-branching structure of many unweighted stock graphs, i.e. the maximum edge-distance between any two nodes is generally short. This violates many prior graph assumptions which are inherent to existing pooling methods. For example, the link prediction loss utilizes the assumption that nearby nodes should be grouped together; however, in a stock graph, edge-distance between nodes are relatively short, thus this additional auxiliary loss is not likely to improve the overall convergence result.

To this end, we introduce the *StockPool* graph pooling operator, which utilizes domain-aware prior assumptions which are specifically tailored for the stock graph. Comprehensive analysis and simulation case studies conclusively demonstrate the computational and performance advantages of the proposed pooling operator against other pooling operators such as DiffPool and MinCUTPool.

6.2 Preliminaries

6.2.1 *Graph Representation of the Stock Market*

In the case of a stock graph, nodes can represent individual stock with the edges as the corresponding relative distances between stocks. We employ the dynamic time warping (DTW) distance [111, 112] as a metric to measure relative distances between the individual stocks.

The problem of stock forecasting is a time series problem, and thus the stock graph must encapsulate a time series element. For simplicity, we employ the log-returns of the past N trading day as the node embedding. In particular, $N = 10$ days is shown to attain the best overall performance amongst the considered GNN models, while log-returns are used due to their approximate Gaussian property [113].

6.2.2 *Graph Convolutional Network*

The first-order approximation of spectral convolution on graphs forms the basis of GCNs [114]. Consider a graph signal, $\boldsymbol{x} \in \mathbb{R}^N$, the forward pass of a GCN computes

$$\boldsymbol{y} = \theta(\boldsymbol{I}_n + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}})\boldsymbol{x} \tag{6.1}$$

where $\boldsymbol{A}$ is the graph adjacency matrix, $\boldsymbol{D}$ is the graph degree matrix, and θ is a learnable parameter. Generally, k convolutional filters can be used, while instead of the graph signal, $\boldsymbol{x}$, it is possible to employ a feature embedding matrix, $\boldsymbol{X} \in \mathbb{R}^{N\times M}$ (i.e. each node carries a signal vector of size M). This modification yields

$$\boldsymbol{Y} = (\boldsymbol{I}_n + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}})\boldsymbol{X}\boldsymbol{\Theta} \tag{6.2}$$

where $\boldsymbol{\Theta} \in \mathbb{R}^{M\times K}$ represents the set of trainable convolution filter weights, and $\boldsymbol{Y} \in \mathbb{R}^{N\times K}$ the output feature embedding matrix of the convolution operation. To prevent instability during the graph convolution process, [114] further proposes a modification in the form of a normalization of (6.2), based on

$$\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_n, \;\; \hat{\boldsymbol{D}}_{ii} = \sum_{j=0} \hat{\boldsymbol{A}}_{ij} \tag{6.3}$$

to yield

$$\boldsymbol{Y} = \hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{X}\boldsymbol{\Theta} \tag{6.4}$$

6.2.3 *Select-Reduce-Connect Framework for Graph Pooling*

Graph pooling can be employed in conjunction with GCN layers to extract high-level features within the graph structure [110]. Denote by $\text{pool}^{(l)}(\cdot)$, $\boldsymbol{H}^{(l)}$ and $\boldsymbol{\Theta}^{(l)}$, the pooling, feature embedding matrix, and the trainable convolutional filter weights at a layer l of the GCN, respectively. Further, denote by $\sigma(\cdot)$ a non-linear activation function. The single-layer update of a GCN with a convolution and pooling layer is then

$$\boldsymbol{H}^{(l+1)} = \text{pool}^{(l)}\left(\text{GCN}\left(\boldsymbol{H}^{(l)}\right)\right) = \text{pool}^{(l)}\left(\sigma\left(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}^{(l)}\boldsymbol{\Theta}^{(l)}\right)\right) \tag{6.5}$$

Given that the pooling operation performs effectively a downsampling of the original graph, we have $\boldsymbol{H}^{(l)} \in \mathbb{R}^{N \times M}$ and $\boldsymbol{H}^{(l+1)} \in \mathbb{R}^{N' \times M'}$, where $N' < N$ (i.e. number of nodes of the coarsened graph is reduced). Different pooling operations can be described using the SRC framework [106]. With $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$ and $\mathcal{G}' = \{\mathcal{N}', \mathcal{E}'\}$ as the original and coarsened graph respectively, the pooling operation can be denoted by $\mathcal{G}' = \text{pool}(\mathcal{G})$. Recall that the SRC framework formalizes graph pooling as a sequence of three distinct steps, the select, reduce, and connect steps. For graphs $\mathcal{G}$ and $\mathcal{G}'$, with the respective number of nodes N and N', the SRC framework can be described as: (i) select the subset of nodes to form a super-node; (ii) reduce the embeddings from the subset of nodes to form the super-node embedding; (iii) connect the super-nodes together in the new graph. It is important to note that different pooling operations approach the select, reduce, and connect steps differently.

6.2.4 *Graph Neural Network Architecture*

The GNN architecture proposed in this work involves a sequence of GCN and hierarchical pooling layers, whereby each layer progressively captures more and more high-level features of the stock market, based on the graph coarsening process. The output embedding of the architecture is fed into a Dense Neural Network (DNN) to perform multi-stock prediction.

6.3 The Stock Pooling Operator

The non-branching nature of stock graphs makes many existing assumptions, that are routinely incorporated into trainable pooling operations (through link prediction loss, mincut loss, etc.), inadequate. To this end, we propose a finance-specific pooling operation, referred to as the *StockPool*, which incorporates the domain knowledge specific to the stock market to achieve an improved end-to-end performance. In particular, StockPool leverages the assumption that stocks within the same industries and sub-industries share similar underlying economic features, thereby allowing for the formation of natural clustering between such stocks. This implies that the cluster assignment of the StockPool operation is deterministic but domain-aware, thus reducing the overall computational cost.

Consider a graph $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$ with N stocks, N' sub-industries, and N'' main industries. The StockPool operation represents a two-step pooling procedure which comprises two hard-cluster assignment (or downsampling) matrices $\boldsymbol{S}^{(1)} \in \mathbb{R}^{N \times N'}$ and $\boldsymbol{S}^{(2)} \in \mathbb{R}^{n' \times n''}$. Denote by $\text{sub}(i)$ a many-to-one mapping of a stock i to its corresponding sub-industry, and by $\text{ind}(j)$ a similar mapping of a sub-industry j to its corresponding main industry. The hard-cluster assignment matrices can then be defined as

$$\boldsymbol{S}^{(1)}{}_{i,j} = \begin{cases} 1, & \text{if } sub(i) = j \\ 0, & \text{otherwise} \end{cases} \tag{6.6}$$

$$\boldsymbol{S}^{(2)}{}_{j,k} = \begin{cases} 1, & \text{if } ind(j) = k \\ 0, & \text{otherwise} \end{cases} \tag{6.7}$$

Figure 6.1 illustrates the deterministic clustering assignment of the StockPool operation, which clusters individual stock nodes to form the sub-industry graph, prior to performing a final clustering of sub-industry nodes to form the sector graph.

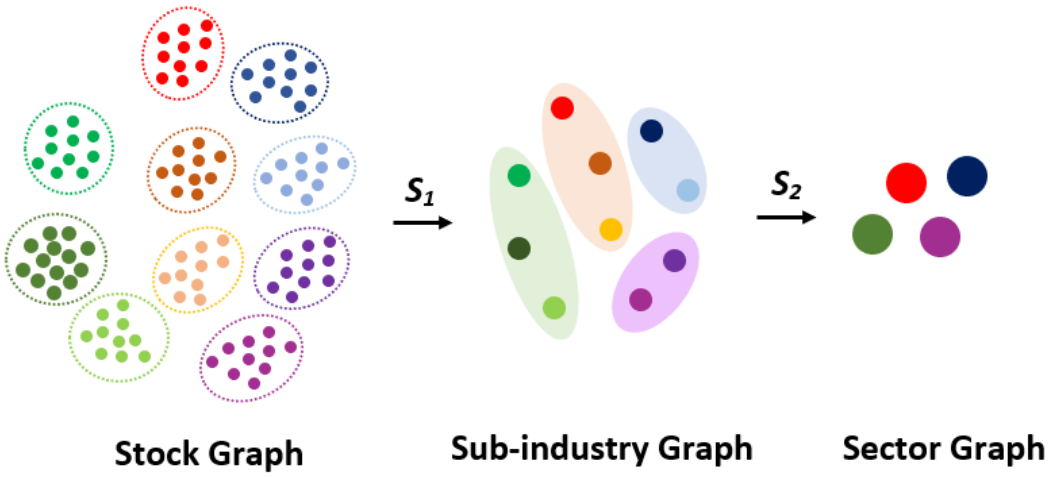


Figure 6.1: Illustration of Two-Layer hierarchical *StockPool*.

Upon employing the hard-clustering assignment matrices in (6.6) and (6.7), the corresponding reduced feature embedding matrix, $\boldsymbol{X}'$, can be computed through a sub-GNN model as

$$\boldsymbol{X}' = \text{pool}\left(\text{GCN}(\boldsymbol{A}, \boldsymbol{X})\right) = \boldsymbol{S}^T \text{GCN}(\boldsymbol{A}, \boldsymbol{X}) \tag{6.8}$$

The reduced adjacency matrix, $\boldsymbol{A}'$, of the coarsened graph can be similarly obtained from the hard-clustering assignment matrices, to yield

$$\boldsymbol{A}' = \boldsymbol{S}^\top \boldsymbol{A} \boldsymbol{S} \tag{6.9}$$

6.4 Graph Ensemble Learning

Ensemble methods employ a set of individually trained weak learners whose predictions are combined to obtain improved generalization performance [115, 116, 117]. While several methods have been proposed to incorporate ensemble learning into graph models [118, 119], the area is still under development. The simulations on the stock market suggest that GNN models with different graph pooling operators tend to exhibit a degree of uncorrelatedness. This desirable property motivates us to exploit a graph ensemble strategy, whereby GNNs with different graph pooling operators serve as a basis for a graph ensemble model. In particular, the StockPool, DiffPool, and MinCUTPool operators were employed to form the graph ensemble by output averaging, as shown in Figure 6.2.

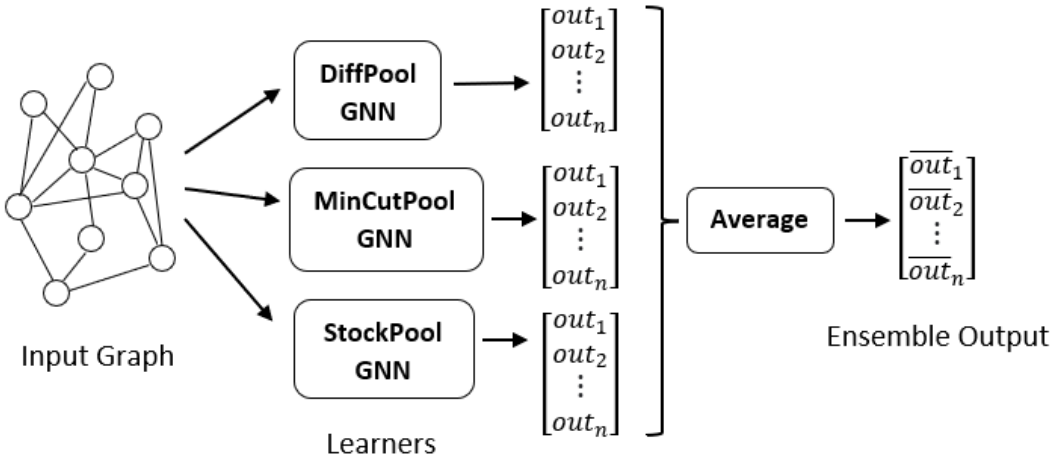


Figure 6.2: A graph ensemble architecture, comprising DiffPool, MinCUTPool, and StockPool GNN.

6.5 Simulations

Performance of the individual and ensemble GNN models was investigated on 457 stocks within the S&P500 index between 2015-01-01 and 2022-01-30. The dataset consists of 11 GICS (Global Industry Classification Standard) industries, which was partitioned into the train and test sets with an 80-20 train-test split; this gave 663,294 training samples and 165,710 test samples.

Model	Test Accuracy(%)	Sharpe Ratio
S&P500 Market (Long)	52.18	1.2613
SAGPool	52.06 (0.0010)	0.9569 (0.16)
TopKPool	52.15 (0.0002)	1.1240 (0.13)
MinCutPool	52.21 (0.0022)	1.3363 (0.23)
DiffPool	52.37 (0.0044)	1.6160 (0.44)
StockPool	**52.57 (0.0021)**	**1.7222 (0.63)**
Ensemble	**52.61 (0.0021)**	**1.7306 (0.27)**

Table 6.1: Test accuracy and Sharpe ratio, evaluated as Mean (standard deviation), of the GNN models and the base S&P500 market.

Evaluation of the models was performed using the binary accuracy and the Sharpe ratio; the mean and standard deviation was calculated over 10 experiments for each model. The Sharpe ratio was computed through a backtesting simulation on the test set, assuming an unweighted portfolio of all 457 stocks. Further, the performance of each model was benchmarked against the base S&P500 market movement (i.e. by taking a long position on the unweighted portfolio), as shown in Table 6.1. Overall, the ensemble model attained the best mean performance in terms of both test accuracy and the Sharpe ratio - in addition to having a reduced variance relative to its constituent sub-models. Figure 6.3 compares the backtesting simulation of the three top performing models, against the ensemble model.

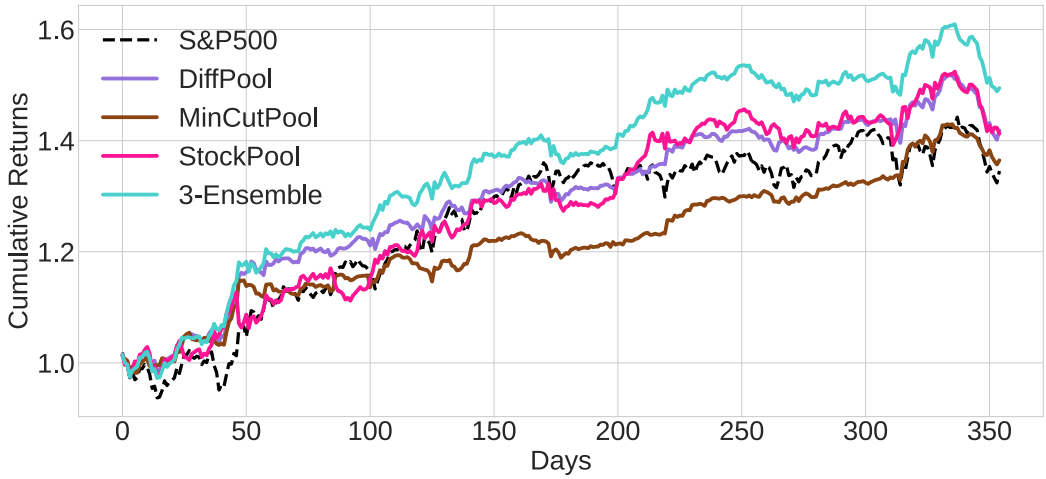


Figure 6.3: Evaluation of GNN models against S&P500 market.

6.6 Summary

We have introduced a finance-specific graph pooling operator, referred as the *StockPool*, with the aim to improve computational efficiency while maintaining enhanced end-to-end performance, compared to existing pooling operations. This has been achieved by incorporating domain knowledge related to the stock graph into the graph coarsening process. Such a domain-aware graph pooling operator has been developed by clustering stocks into their corresponding industries, under the assumption that such stocks share key underlying economic features. We have also investigated the use of different graph pooling operators as a basis for constructing highly uncorrelated GNN models, a pre-requisite for a successful ensemble model. Experimentation based on the S&P500 stock index have demonstrated that the graph ensemble model outperforms its constituent sub-models, as well as the general movement of the market.

Part III

Tensor-Variate Machine Learning on Graphs

Chapter 7

Graph-Regularized Tensor Regression

Part III of this thesis begins by examining the application of graph-theoretic concepts to enhance tensor models. This is increasingly essential due to the exceedingly large and multi-dimensional data sources that define modern big data. This represents a challenge for traditional ML models, as the number of model parameters needed to process such data grows exponentially with the data dimensions; an effect known as the curse of dimensionality. Recently, TD techniques have shown promising results in reducing the computational costs associated with large-dimensional models while achieving comparable performance. However, such tensor models are often unable to incorporate the underlying domain knowledge when compressing high-dimensional models. To this end, we introduce a novel GRTR framework, whereby domain knowledge about intra-modal relations is incorporated into the model in the form of a graph Laplacian matrix. This is then used as a regularization tool to promote a physically meaningful structure within the model parameters. By virtue of tensor algebra, the proposed framework is shown to be fully interpretable, both coefficient-wise and dimension-wise. The GRTR model is validated in a multi-way regression setting and compared against competing models, and is shown to achieve improved performance at reduced computational costs. Detailed visualizations are provided to help the reader gain an intuitive understanding of the employed tensor operations.

7.1 Introduction

The modern big data is naturally large- and multi-dimensional; indeed, a typical data setting involves high-resolution data collected over time and across many sensors, leading to an explosion in dimensionality. This results in excessively large number of model parameters which renders the training prone to over-fitting and impedes efficient learning, a phenomenon known as the curse of dimensionality [17].

Tensor-based models [83, 13, 16] are an emerging framework which is naturally suited to deal with the curse of dimensionality. Tensors are a multi-dimensional generalization of vectors (order-1 tensors) and matrices (order-2 tensors). In particular, TD techniques, similarly to their matrix decomposition counterparts, have the ability to logarithmically compress large multi-dimensional tensors into an efficient low-rank format by exploiting intra- and inter-dependencies across data dimensions

(tensor modes) [18, 19, 20]. This both preserves the underlying data structure and interpretability, while bypassing the curse of dimensionality, as the associated computational costs reduce from an original exponential one to a linear one in the data dimensions. Several TD techniques, such as CPD [23, 24], TKD [25, 26], and TTD [27], have been successfully applied to develop regression models for tensor-variate data.

However, modern data also often exhibits irregular relational structures, which cannot be handled via traditional ML techniques. For instance, pairs of sensors can exhibit varying degrees of correlations, depending on their physical proximity. Such irregular relational structures cannot be straightforwardly integrated into standard tensor methods, resulting in sub-optimal results. To this end, GSP techniques have recently attracted the interest of the data analytics community, as they generalize traditional ML techniques to data on irregular domains [35, 34, 36, 110], achieving state-of-the-art performance in tasks where the underlying data relations can be captured as a graph. For instance, in financial modelling [16, 120], assets can be represented as nodes of a graph, while the edges connecting different nodes may be weighted depending on their relational information.

The modern big data is both large- and multi-dimensional, while simultaneously residing on irregular domains, which poses significant challenges to traditional ML methods. For instance, analysis of foreign exchange data [16] requires the processing of currency information across time, countries, and market features (order-3 financial tensor), while accounting for the underlying cross-country interest rate differences (financial graph). Similarly, analysis of surveillance camera data for traffic monitoring [59] requires the processing of video data (order-4 video tensor), while accounting for the underlying road connectivity (traffic graph).

Analytics of modern big data would therefore greatly benefit from a joint consideration of tensor and graph techniques, to simultaneously overcome the curse of dimensionality, by virtue of tensors, while benefiting from domain knowledge about the underlying relational structures, through graphs. To this end, we introduce a novel GRTR framework, which allows for the regression of large multi-dimensional data while efficiently incorporating prior domain knowledge into the model parameter estimation. More specifically, within the proposed model, intra-modal knowledge about tensor modes is encoded as a graph Laplacian matrix. This is then used to regularize the estimation of TR parameters, which are stored in a low-rank TD format. In addition to improved performance and reduced space complexity, such a model is fully interpretable, both in the coefficient-wise and mode-wise sense. Particular effort is made to provide intuitive understanding of the employed techniques through detailed visualizations of the underlying operations using TN diagrams, giving readers with an insight into the advantages of tensor algebra when dealing with multiple data dimensions.

The rest of the chapter is organized as follows. Related work is first discussed in Section 7.2. We introduce the proposed GRTR framework, derive the corresponding learning algorithm, and analyse the associated space and time complexities in Section 7.3. The proposed model is validated and compared against traditional algorithms in a synthetic data based experiment in Section 7.4, as well as in a real data based

regression experiment in Section 7.5. The interpretability of the trained model is then discussed in detail in Section 7.6. Finally, we conclude and summarize our findings in Section 7.7.

7.2 Related Work

The proposed GRTR framework can be considered under the CPD [44] TR paradigm [24, 23, 121]. Further work on TR using other TD techniques includes [25] and [26], where the authors assume respectively a Tucker and a HT format of the regression coefficients. Furthermore, generalized tensor-on-tensor regression was introduced using the TT format in [27]. Bayesian TR was proposed in [122]. However, contrary to the above TR methods, the proposed graph regularization makes it possible to incorporate domain knowledge about intra-modal relations. This is different from graph-regularized TD models [123, 124, 125], which have been used to only help preserve local relationships among tensor samples. In other words, whereas such models are concerned with tensor representations of multi-dimensional data, we are concerned with regression tasks. The recently introduced GTN framework [10, 11, 16] also employs graphs within the TN paradigm. However, it is a black-box model with graphs serving as an added low-pass filter, whereas the present work employs graph regularization as a mathematical constraint. This allows us to maintain model interpretability throughout the data processing chain, while identifying and promoting meaningful relationships between the regression variables. The proposed GRTR framework is also different from TD based kernel regression methods [126, 127], as hand-crafted feature mappings are not required. Further work on high-dimensional model compression methods include [75, 74, 28, 128], where the authors employ TD techniques to heavily compress fully-connected, convolutional, and RNNs, without a significant loss in performance while enjoying great speedups in training and inference time. However, such methods do not allow for the incorporation of domain knowledge.

7.3 Graph-Regularized Tensor Regression

7.3.1 *Tensor Regression*

For a set of tensor inputs, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, and a set of corresponding output labels, $y \in \mathbb{R}$, the TR [23] can be expressed as

$$y = \langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}} \rangle + b. \tag{7.1}$$

Analogously to standard regression, TR aims to learn: (i) the regression constant, $b \in \mathbb{R}$, and (ii) the weight tensor, $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, in the CPD format (see Section 9), such that the prediction mean-squared-error (MSE) in (7.1) is minimized for a given dataset.

Remark 10 *Instead of directly learning the large weight tensor, $\mathcal{W}$, TR learns its low-rank CPD factor matrices, $\boldsymbol{U}^{(n)}$ (one for each of the N input dimensions), such that equation (2.9) holds. The TR framework in (7.1) is illustrated in Figure 7.1 in the TN notation.*

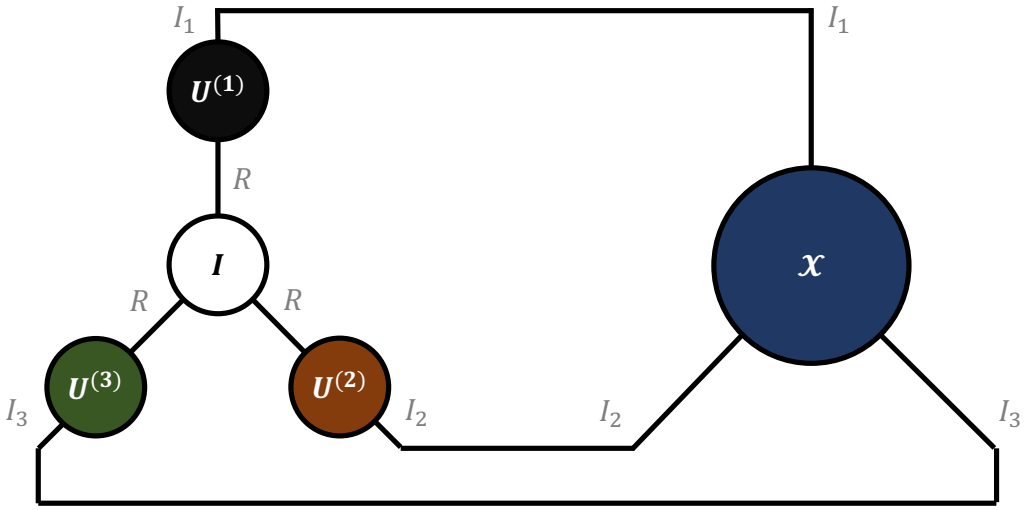


Figure 7.1: TN diagram of the inner product, $\langle \mathcal{W}, \mathcal{X} \rangle$, within an order-3 TR. The weight tensor, $\mathcal{W}$, on the left is factorized by the CPD according to equation (2.9). The connected tensor modes represent tensor contractions along these modes.

The learning of the CPD factor matrices is often achieved through the ALS procedure [23], which iteratively updates each factor matrix by minimizing the MSE with respect to mode-n, $n = 1, \ldots, N$, until convergence.

Remark 11 *For a learned weight tensor, $\mathcal{W}$, the TR framework in (7.1) is mathematically equivalent to a classical Linear Regression (LR), $y = \boldsymbol{w}^T \boldsymbol{x} + b$, where $\boldsymbol{w}$ and $\boldsymbol{x}$ are respectively the vectorized versions of $\mathcal{W}$ and $\mathcal{X}$, given by*

$$y = \langle \mathcal{X}, \mathcal{W} \rangle + b = \text{vec}(\mathcal{X})^T \, \text{vec}(\mathcal{W}) + b = \boldsymbol{x}^T \boldsymbol{w} + b. \tag{7.2}$$

The main difference (and advantage) of the TR framework over a standard multivariate regression approach is that it maintains the multi-dimensional structure of the original data, allowing us to exploit its multi-way, low-rank CPD structure during the training (i.e., parameter estimation) stage. This reduces both the computational complexity and the likelihood of over-fitting, while maintaining the interpretability throughout the processing chain.

7.3.2 *Graph-Regularized Tensor Regression*

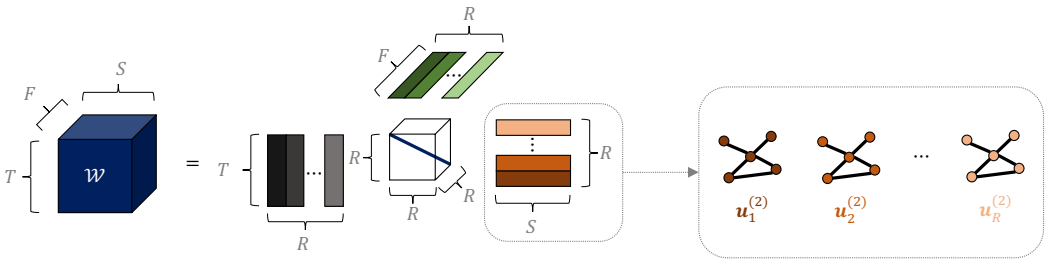


Figure 7.2: Principle of GRTR. The order-3 weight tensor, $\mathcal{W} \in \mathbb{R}^{T \times S \times F}$, is represented as a rank-R CPD TD, where the corresponding factor matrices are of dimensions, $\boldsymbol{U}^{(1)} \in \mathbb{R}^{T \times R}$, $\boldsymbol{U}^{(2)} \in \mathbb{R}^{S \times R}$, and $\boldsymbol{U}^{(3)} \in \mathbb{R}^{F \times R}$. Our aim is to regularize the learning of the mode-2 factor matrix, $\boldsymbol{U}^{(2)} \in \mathbb{R}^{S \times R}$, such that it reflects the physical proximity information of the S sensors as domain knowledge. To this end, we construct a graph with S vertices representing S sensors, where an edge connects a pair of vertices with an edge weight proportional to their physical proximity. Given such a mode-2 graph, the corresponding factor matrix, $\boldsymbol{U} = [\boldsymbol{u}_1^{(n)}, \ldots, \boldsymbol{u}_R^{(n)}]$, can be viewed as a set of R graph signals (its column vectors), which can be learned so as to be smooth with respect to the underlying sensors graph. This makes it possible to regularize the learning of the factor matrix and results, as desired, in similar weights for neighbouring sensors.

In practice, when learning the factor matrices, $\boldsymbol{U}^{(n)} \in \mathbb{R}^{I_n \times R}$, in addition to minimizing the MSE, the parameter estimation is often regularized by the L_2-norm of the factor matrices to promote smaller norms. However, such a regularization scheme is ignorant to the intra-modal relations present in the data. In other words, given some domain knowledge about the relational structure within mode-n (e.g., physical proximity among I_n sensors), a naive L_2-regularization scheme cannot incorporate such domain knowledge into the model training process.

To this end, we propose an extension to the classical TR paradigm to enable the incorporation of domain knowledge into the learning process. Given a graph Laplacian matrix, $\boldsymbol{L}^{(n)} \in \mathbb{R}^{I_n \times I_n}$, which encodes domain knowledge about the relational structure within mode-n, we aid the learning of the mode-n factor matrix, $\boldsymbol{U}^{(n)}$, through a corresponding mode-n graph Laplacian regularization, as discussed in Section 2.2. In this way, domain-knowledge about data modes informs the model, which reduces the likelihood of over-fitting and improves the learning process.

Remark 12 *The column vectors of the mode-n CPD factor matrix, $\boldsymbol{u}_r^{(n)} \in \mathbb{R}^{I_n}$ for $r = 1, \ldots, R$, can be viewed as graph signals. In this way, the estimation of the factor matrix can be interpreted as the estimation of R graph signals that are smooth with respect to the graph topology encoded by the mode-n Laplacian matrix, $\boldsymbol{L}^{(n)} \in \mathbb{R}^{I_n \times I_n}$.*

Example 3 *Consider an order-3 input tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{T \times S \times F}$, where F features for S sensors are indexed over T time-steps. For this data, we wish to learn a TR model, such that the mode-2 (the sensors mode) factor matrix, $\boldsymbol{U}^{(2)} \in \mathbb{R}^{S \times R}$, incorporates the physical proximity knowledge of the S sensors, as illustrated in Figure 7.2. To*

this end, we construct a graph with S vertices representing S different sensors, where the edges connect a pair of sensors with edge weights proportional to their proximity. Given this graph, the column vectors of the mode-2 factor matrix, $\boldsymbol{u}_1^{(n)}, \ldots, \boldsymbol{u}_R^{(n)}$, can be interpreted as a set of R graph signals. Therefore, the learning of the factor matrix, $\boldsymbol{U}^{(2)}$, can be regularised by ensuring that its column vectors, $\boldsymbol{u}_r^{(2)}$, $r = 1, \ldots, R$, are smooth with respect to the underlying sensor graph (i.e., the graph signal smoothness, $s = \operatorname{tr}\left(\boldsymbol{U}^{(2)^T} \boldsymbol{L}^{(2)} \boldsymbol{U}^{(2)}\right)$, is small).

To formally introduce the concept of graph regularization in the classical TR framework, we define a mode-n graph Laplacian matrix, $\boldsymbol{L}^{(n)}$, for each of the N tensor modes, and assign a corresponding regularization constant, $\lambda^{(n)}$. The mode-wise graph Laplacian regularization terms are then included into the MSE loss function, calculated over $m = 1, \ldots, M$ samples, to yield the following loss function to be minimized

$$\begin{aligned} \mathcal{L} = & \underbrace{\frac{1}{M} \sum_{m=1}^{M} \frac{1}{2} \left(y^{(m)} - \left\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}^{(m)} \right\rangle - b \right)^2}_{\text{Error term}} \\ & + \underbrace{\frac{1}{M} \sum_{n=1}^{N} \frac{1}{2} \left(\lambda^{(n)} \operatorname{tr}\left(\boldsymbol{U}^{(n)^T} \boldsymbol{L}^{(n)} \boldsymbol{U}^{(n)} \right) \right)}_{\text{Regularization term}}. \end{aligned} \tag{7.3}$$

The constant, $b \in \mathbb{R}$, that minimizes the loss function, $\mathcal{L}$, is estimated based on the derivative of the loss function with respect to this constant using the chain-rule, resulting in

$$\frac{\partial \mathcal{L}}{\partial b} = -\frac{1}{M} \sum_{m=1}^{M} \left(y^{(m)} - \left\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}^{(m)} \right\rangle - b \right). \tag{7.4}$$

By setting the partial derivative in (7.4) to zero, the constant term update can be obtained in closed form as

$$b = \frac{1}{M} \sum_{m=1}^{M} \left(y^{(m)} - \left\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}^{(m)} \right\rangle \right). \tag{7.5}$$

The factor matrices that minimize the loss function are estimated by the ALS framework [129], which iteratively minimizes the loss function one-mode-at-a-time. To this end, we apply the property in (2.10) and matricize the inner product with respect to the n-th mode as

$$\begin{aligned} \left\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}^{(m)} \right\rangle &= \left\langle \boldsymbol{W}_{\{n\}}, \boldsymbol{X}_{\{n\}}^{(m)} \right\rangle \\ &= \left\langle \boldsymbol{U}^{(n)} \boldsymbol{U}^{(-n)^T}, \boldsymbol{X}_{\{n\}}^{(m)} \right\rangle \\ &= \operatorname{tr}\left(\boldsymbol{U}^{(n)} \boldsymbol{U}^{(-n)^T} \boldsymbol{X}_{\{n\}}^{(m)^T} \right) \\ &= \operatorname{tr}\left(\boldsymbol{U}^{(n)} \tilde{\boldsymbol{X}}_{\{n\}}^{(m)^T} \right), \end{aligned} \tag{7.6}$$

where $\boldsymbol{U}^{(-n)} = \left(\bigodot_{i=N,i\neq n}^{1} \boldsymbol{U}^{(i)}\right)$, $\boldsymbol{X}_{\{n\}}^{(m)^T}$ is the mode-n matricization of the m-th input tensor sample, and $\tilde{\boldsymbol{X}}_{\{n\}}^{(m)} = \boldsymbol{X}_{\{n\}}^{(m)}\boldsymbol{U}^{(-n)}$. This mode-$n$ matricized formulation allows for the computation of the partial derivative with respect to mode-n factor matrix, by leveraging on the well-known matrix derivatives rules

$$\frac{\partial}{\partial \boldsymbol{U}^{(n)}} \operatorname{tr}\left(\boldsymbol{U}^{(n)}\tilde{\boldsymbol{X}}_{\{n\}}^{(m)^T}\right) = \tilde{\boldsymbol{X}}_{\{n\}}^{(m)}. \tag{7.7}$$

Through the chain rule, the above results can be included into the calculation of the derivative of the loss function with respect to the mode-n factor matrix as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{U}^{(n)}} = &-\frac{1}{M}\sum_{m=1}^{M}\left(y^{(m)} - \operatorname{tr}\left(\boldsymbol{U}^{(n)}\tilde{\boldsymbol{X}}_{\{n\}}^{(m)^T}\right) - b\right)\tilde{\boldsymbol{X}}_{\{n\}}^{(m)} \\ &+ \frac{1}{M}\lambda^{(n)}\boldsymbol{L}^{(n)}\boldsymbol{U}^{(n)}\end{aligned} \tag{7.8}$$

By setting the partial derivative in (7.8) to zero, the terms in (7.8) can be rearranged as

$$\begin{aligned}\sum_{m=1}^{M}\left(y^{(m)} - b\right)\tilde{\boldsymbol{X}}_{\{n\}}^{(m)} = &\sum_{m=1}^{M}\operatorname{tr}\left(\boldsymbol{U}^{(n)}\tilde{\boldsymbol{X}}_{\{n\}}^{(m)^T}\right)\tilde{\boldsymbol{X}}_{\{n\}}^{(m)} \\ &+ \lambda^{(n)}\boldsymbol{L}^{(n)}\boldsymbol{U}^{(n)}.\end{aligned} \tag{7.9}$$

The left-hand side of (7.9) can be simplified though the vectorization operation as

$$\sum_{m=1}^{M}\left(y^{(m)} - b\right)\operatorname{vec}\left(\tilde{\boldsymbol{X}}_{\{n\}}^{(m)}\right) = \boldsymbol{\Psi}^{(n)}\boldsymbol{\gamma}, \tag{7.10}$$

where $\boldsymbol{\Psi}^{(n)} \in \mathbb{R}^{M\times I_n R}$ is a matrix obtained by stacking $\operatorname{vec}\left(\tilde{\boldsymbol{X}}_{\{n\}}^{(m)}\right)$ as row vectors across M samples, and $\boldsymbol{\gamma} \in \mathbb{R}^{M}$ is a vector such that $\boldsymbol{\gamma}_m = y^{(m)} - b$.

Similarly, the right-hand side of (7.9) can also be simplified though the vectorization operation as

$$\begin{aligned}&\sum_{m=1}^{M}\operatorname{tr}\left(\boldsymbol{U}^{(n)}\tilde{\boldsymbol{X}}_{\{n\}}^{(m)^T}\right)\tilde{\boldsymbol{X}}_{\{n\}}^{(m)} + \lambda^{(n)}\boldsymbol{L}^{(n)}\boldsymbol{U}^{(n)} \\ &= \boldsymbol{\Psi}^{(n)^T}\left(\boldsymbol{\Psi}^{(n)}\boldsymbol{u}^{(n)}\right) + \lambda^{(n)}\left(\boldsymbol{L}^{(n)}\otimes\boldsymbol{I}\right)\boldsymbol{u}^{(n)} \\ &= \left(\boldsymbol{\Psi}^{(n)^T}\boldsymbol{\Psi}^{(n)} + \lambda^{(n)}\left(\boldsymbol{L}^{(n)}\otimes\boldsymbol{I}\right)\right)\boldsymbol{u}^{(n)},\end{aligned} \tag{7.11}$$

where $\boldsymbol{u}^{(n)} = \operatorname{vec}\left(\boldsymbol{U}^{(n)}\right) \in \mathbb{R}^{I_n R}$.

By combining (7.10) and (7.11), the mode-n factor matrix update can be obtained in a closed-form as

$$\boldsymbol{u}^{(n)} = \left(\boldsymbol{\Psi}^{(n)^T}\boldsymbol{\Psi}^{(n)} + \lambda^{(n)}\left(\boldsymbol{L}^{(n)}\otimes\boldsymbol{I}\right)\right)^{-1}\boldsymbol{\Psi}^{(n)}\boldsymbol{\gamma}. \tag{7.12}$$

The resulting vector, $\boldsymbol{u}^{(n)}$, can be simply matricized back to obtain $\boldsymbol{U}^{(n)}$.

Remark 13 *Equation (7.12) considers the general case where each mode is associated with an underlying graph domain. However, if the n-th mode does not have an associated Laplacian matrix, then the mode-n factor matrix update is simplified to* $\boldsymbol{u}^{(n)} = \left(\boldsymbol{\Psi}^{(n)^T}\boldsymbol{\Psi}^{(n)}\right)^{-1}\boldsymbol{\Psi}^{(n)}\boldsymbol{\gamma}$.

By virtue of multi-linear algebra, the partial derivatives for the constant term and the factor matrices have an exact closed form solution, which can be used to iteratively update the model parameters until convergence. The corresponding learning algorithm is summarized in Algorithm 4.

Algorithm 4: GRTR via ALS

Input: Set of features and labels, $\{\boldsymbol{\mathcal{X}}^{(m)}, y^{(m)}\}$, set of Laplacian matrices, $\{\mathbf{L}^{(n)}\}$, set of regularization constants, $\{\lambda^{(n)}, \rho^{(n)}\}$, model rank, R, model error tolerance, t, maximum steps K.
Output: Tensor weights, $\boldsymbol{U}^{(1)}, \ldots, \boldsymbol{U}^{(N)}$

Initialise $\boldsymbol{U}^{(n)} \in \mathbb{R}^{I_n \times R}$ randomly, $n = 1, \ldots, N$.
Initialise $b \in \mathbb{R}$ randomly.
Initialise $\epsilon = \infty$.
Initialise $k = 0$.

while $\epsilon > t$ *and* $k < K$ **do**
  $\boldsymbol{\mathcal{W}} \leftarrow \boldsymbol{\mathcal{I}} \times_1^1 \boldsymbol{U}^{(1)} \times_2^1 \cdots \times_N^1 \boldsymbol{U}^{(N)}$
  $\epsilon \leftarrow \frac{1}{M}\sum_{m=1}^{M}(y^{(m)} - \left\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}^{(m)} \right\rangle - b)^2$
  for $n = 1, \ldots, N$ **do**
    Update b according to (7.5)
    Update $\boldsymbol{U}^{(n)}$ according to (7.12)
  end
  $k \leftarrow k + 1$
end
Return: $\boldsymbol{U}^{(1)}, \ldots, \boldsymbol{U}^{(N)}$

7.3.3 *Choice of Model Rank*

A key hyper-parameter choice within the GRTR model is the model rank. A larger rank increases the expressive power of the model, but it also increases the space complexity of the model which can lead to over-fitting. In practice, the optimal choice of model rank can be determined via grid-search, whereby the rank with the smallest cross-validation error is used for the final model.

7.3.4 *Choice of Regularization Parameter*

The regularization parameter, $\lambda^{(n)}$, is another important hyper-parameter. As $\lambda^{(n)}$ increases, the regularization term begins to dominate the loss function over the error term in (7.3), which reduces the likelihood of in-sample over-fitting and thus resulting in enhanced out-of-sample performance. In practice, the optimal choice of regularization parameter can also be determined empirically via grid search.

Example 4 *To illustrate the effect of the graph regularization parameter, consider a GRTR model of rank $R = 1$, whereby the mode-n factor matrix, $\boldsymbol{U}^{(n)}$, is reduced to a column vector, $\boldsymbol{u}^{(n)}$. This factor vector is effectively a graph signal, which contains a scalar value for each of the I_n vertices in the mode-n graph implied by the corresponding Laplacian matrix, $\boldsymbol{L}^{(n)}$. For large values of $\lambda^{(n)}$, this leads to highly smooth vectors, $\boldsymbol{u}^{(n)}$, where the difference between entries, $|\boldsymbol{u}_i^{(n)} - \boldsymbol{u}_j^{(n)}|$, is small if the i-th and j-th vertices are closely connected in the underlying mode-n graph. This effect is illustrated in Figure 7.3.*

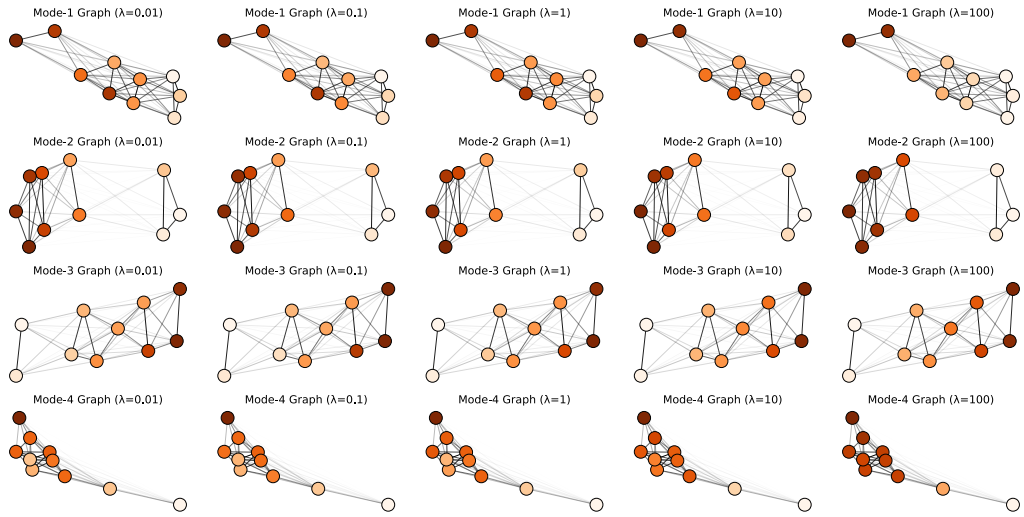


Figure 7.3: Visualization of the effect of the graph regularization parameter, $\lambda^{(n)}$, on the estimated factor vectors, $\boldsymbol{u}^{(n)} \in \mathbb{R}^{I_n}$. The row panels illustrate $N = 4$ different graph domains, each with $I_n = 10$ vertices that are coloured proportionally to the values of the corresponding factor vectors. The column panels illustrate the factor vectors for different values of $\lambda^{(n)}$. Notice how the factor vectors become smoother with respect to the underlying graph with an increase in regularization parameter, as indicated by the neighbouring vertices exhibiting increasingly similar colours with larger $\lambda^{(n)}$.

7.3.5 *Computational Complexity Analysis*

We now proceed to analyse the space complexity (i.e., the number of trainable parameters) and the time complexity (i.e., the number of basic computations) of the

proposed algorithm. For simplicity of analysis, $I_n = I$ for $n = 1, \ldots, N$ and $R \ll I$ is assumed without loss of generality.

Once the model weights are estimated, the full weight tensor, $\boldsymbol{\mathcal{W}}$, can be reconstructed according to (2.8). In this way, as shown in (7.2), the space and time complexity of the inference becomes the same as the standard LR:

- Space complexity: $\mathcal{O}\left(I^N\right)$
- Time complexity: $\mathcal{O}\left(I^N\right)$

Alternatively, to reduce the space complexity, the trained factor matrices, $\boldsymbol{U}^{(n)}$, can be kept in a low-rank CPD format, without reconstructing the full weight tensor, $\boldsymbol{\mathcal{W}}$. However, this would slightly increase the time-complexity of the inference since it would require N tensor contractions as

$$\begin{aligned} y &= \langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{W}} \rangle + b \\ &= \boldsymbol{\mathcal{X}} \times_{1,\ldots,N}^{1,\ldots,N} \left(\boldsymbol{\mathcal{I}} \times_1^1 \boldsymbol{U}^{(1)} \times_2^1 \cdots \times_N^1 \boldsymbol{U}^{(N)} \right) + b \\ &= \left(\boldsymbol{\mathcal{X}} \times_1^1 \boldsymbol{U}^{(1)} \times_2^1 \cdots \times_N^1 \boldsymbol{U}^{(N)} \right) \times_{1,\ldots,N}^{1,\ldots,N} \boldsymbol{\mathcal{I}} + b. \end{aligned} \tag{7.13}$$

Therefore, by maintaining the low-rank form, the space complexity can be reduced from an exponential one to a linear one in the tensor order, N, by sacrificing time complexity slightly:

- Space complexity: $\mathcal{O}\left(NIR\right)$
- Time complexity: $\mathcal{O}\left(RI^N\right) \approx \mathcal{O}\left(I^N\right)$

The derived space and time complexities were validated through simulations using a TR model with $I = 2$ and $R = 3$. The results are illustrated in Figure 7.4.

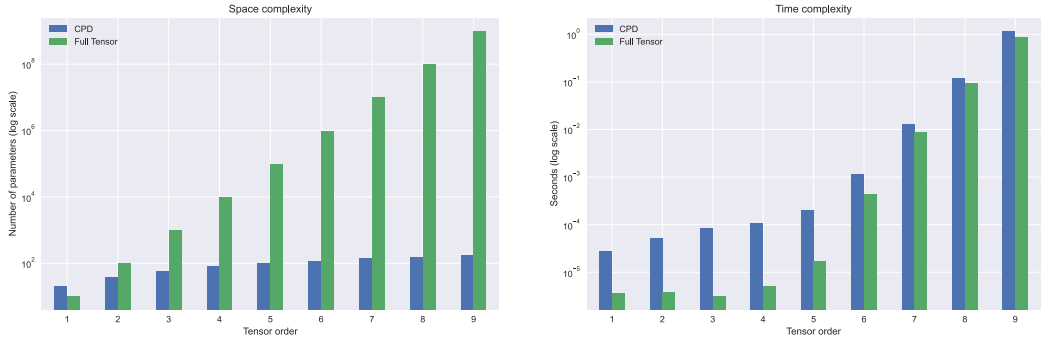


Figure 7.4: Space (left) and time (right) complexity simulations for a TR model of rank $R = 3$. The space complexity (i.e., number of parameters) is linear in the tensor order for the low-rank form, $\mathcal{O}\left(NIR\right)$, but exponential for the full tensor form, $\mathcal{O}\left(I^N\right)$. The time complexity (in seconds) is exponential in the tensor order for both the low-rank form, $\mathcal{O}\left(RI^N\right)$, and the full tensor form, $\mathcal{O}\left(I^N\right)$, which are asymptotically the same for large N.

7.4 Experiments on Synthetic Data

To validate the ability of the proposed model to capture inter- and intra-modal relations in data, as opposed to classical models, we conducted an experiment on synthetic data involving large and noisy tensor data. For reproducibility, the experimental code can be found on GitHub[1].

7.4.1 *Experimental Setting*

Consider a regression problem which is fully determined by an unknown CPD weight tensor, $\boldsymbol{\mathcal{W}}$. This system generates a set of M observations, $\left(\boldsymbol{\mathcal{X}}^{(m)}, y^{(m)}\right)$, such that $y^{(m)} = \langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}^{(m)} \rangle + \eta^{(m)}$, where η is additive random noise. Given this set of noisy observations, the goal of the experiment is to estimate the ground-truth weight tensor, $\tilde{\boldsymbol{\mathcal{W}}}$, using regression methods, such that the MSE, $||\tilde{\boldsymbol{\mathcal{W}}} - \boldsymbol{\mathcal{W}}||_2^2$, is as small as possible.

7.4.2 *Data*

To carry out the proposed regression experiment, $N = 4$ ground-truth factor matrices, $\boldsymbol{U}^{(n)} \in \mathbb{R}^{10\times 5}$, were sampled randomly from a uniform distribution with rank $R = 5$ and dimensions $I_n = 10$, $n = 1, \ldots, 4$. The factor matrices were then contracted according to (2.9) to generate the ground-truth weight tensor of order-4, $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{10\times 10\times 10\times 10}$. A set of $M = 125$ input tensors, $\boldsymbol{\mathcal{X}}^{(m)} \in \mathbb{R}^{10\times 10\times 10\times 10}$, were then sampled randomly from a Gaussian distribution with zero-mean and unit-variance, which were then contracted with the weight tensor to generate a corresponding set of regression labels, $y^{(m)} = \langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}^{(m)} \rangle + \eta^{(m)}$, where $\eta^{(m)}$ is additive noise sampled from a Gaussian distribution with zero-mean and variance, σ_η^2, such that $\sigma_\eta = 0.5\sigma_y$, where σ_y is the standard deviation of the regression labels. The $M = 125$ observations were then split according to a standard $80\% - 20\%$ train-test split. Figure 7.5 illustrates the proposed experimental setting.

Remark 14 *The input tensor samples, $\boldsymbol{\mathcal{X}}^{(m)} \in \mathbb{R}^{10\times 10\times 10\times 10}$, are order-$4$ tensors with mode size $I_n = 10$, which effectively contains $10^4 = 10,000$ features for every sample. However, only 100 samples were available for the training set, which is 2 orders of magnitude smaller than the number of features. The curse of dimensionality inherent to this experiment, coupled with a low signal-to-noise ratio of $\frac{\sigma_y}{\sigma_\eta} = 2$ (6 dB), poses significant challenges to traditional models, both in terms of complexity and over-fitting.*

[1] `www.github.com/gylx/tensor_regression_graph`

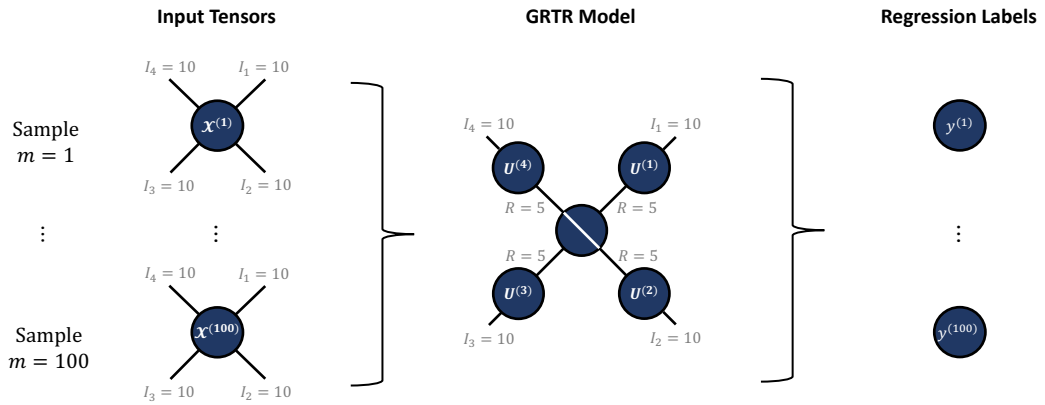


Figure 7.5: Experimental setting of the synthetic data experiment in TN notation. The training set consists of $M = 100$ samples of order-4 input tensors, $\boldsymbol{\mathcal{X}}^{(m)} \in \mathbb{R}^{10\times10\times10\times10}$, and the corresponding noisy regression labels, $y^{(m)}$. The goal of the experiment is to estimate the ground-truth factor matrices, $\boldsymbol{U}^{(n)} \in \mathbb{R}^{10\times5}$, $n = 1, \ldots, 4$, from the training set data.

7.4.3 *Incorporation of Domain Knowledge with Graphs*

The proposed GRTR framework allows for the incorporation of domain knowledge about the underlying ground-truth weight tensor. This is achieved by defining a mode-n graph which captures the corresponding intra-modal structure. For the proposed experiment, the elements of the mode-n graph adjacency matrix, $\boldsymbol{A}^{(n)} \in \mathbb{R}^{10\times10}$, $n = 1, \ldots, 4$, were computed using a Gaussian kernel as

$$\boldsymbol{A}^{(n)}_{i,j} = e^{\left(-\beta||\boldsymbol{u}^{(n)}_i - \boldsymbol{u}^{(n)}_j||^2_2\right)}, \tag{7.14}$$

where $\boldsymbol{u}^{(n)}_i \in \mathbb{R}^5$ and $\boldsymbol{u}^{(n)}_j \in \mathbb{R}^5$ denote respectively the i-th and j-th row vector of the ground-truth mode-n factor matrix, $\boldsymbol{U}^{(n)} \in \mathbb{R}^{10\times5}$. In this way, the entries of the mode-n graph adjacency matrix encode the similarity between row vectors of the ground-truth mode-n factor matrix. The mode-n Laplacian matrices, $\boldsymbol{L}^{(n)} \in \mathbb{R}^{10\times10}$, were then computed as discussed in Section 2.2, which were used to regularize the learning process and to guide the estimated weights closer to the ground-truth.

7.4.4 *Models*

The proposed GRTR was evaluated against: (i) LR, (ii) LR with L2 regularization (L2LR), (iii) standard TR, and (iv) TR with L2 regularization (L2TR). For the LR and L2LR models, the ground-truth weight tensor was vectorized as $\boldsymbol{w} = \text{vec}\,(\boldsymbol{\mathcal{W}}) \in \mathbb{R}^{10^4=10,000}$.

7.4.5 *Results*

Simulation results are illustrated in Table 7.1. Linear models, such as LR and L2LR, needed to estimate 10,000 parameters from 100 noisy samples, which led to over-fitting in the training set due to the noisy and large-dimensional nature of data. This is reflected in the large mean-squared-errors in the estimation of the ground-truth weight tensor. On the other hand, classical tensor models, such as TR and L2TR, reduced the space complexity to only 200 parameters by virtue of their tensor formulation. However, they still struggled to estimate the ground-truth weights due to large noise and small number of samples. The proposed GRTR, aided by the domain knowledge encoded in the mode-n graphs, achieved the best overall scores by a large margin, while incurring the same small complexity cost of other tensor models.

Model	LR	L2LR	TR	L2TR	GRTR
MSE	0.137	0.137	0.207	0.087	**0.079**
Params	10,000	10,000	**200**	**200**	**200**

Table 7.1: Experiment results for the considered models. The first row shows the MSE bewteen the ground-truth weight tensor and the estimated weight tensor. The second row shows the number of parameters as a measure of space complexity. The proposed GRTR model achieved the best results by a large margin while requiring only 200 parameters, while the other considered models over-fit the training set owing to the noisy and large-dimensional nature of the problem.

7.5 Experiments on Real World Data

We next validated the proposed model in a real-world multi-way regression experiment.

7.5.1 *Data*

The Beijing Multi-Site Air Quality dataset [130] was considered for the experiment, owing to its multi-way tensor structure and its irregular sensor placement which naturally admits a graph representation. The dataset consists of 12 air quality variables (`PM2.5`, `PM10`, `SO2`, `NO2`, `CO`, `O3`, `TEMP`, `PRES`, `DEWP`, `RAIN`, `wd`, `WSPM`) recorded hourly across 12 different geographical sites in Beijing. Using a sliding window of 12 time-steps, the multi-site nature of the experiment data naturally leads to tensor-structured inputs for each point in time, $\boldsymbol{X}^{(t)} \in \mathbb{R}^{T\times S\times F}$, where $F = 12$ features for $S = 12$ sites were extracted over $T = 12$ past time-steps. One-hot-encoding was applied to the categorical variable `wd` (wind directions with 16 possible values), resulting in $F = 27$ features in total. For each of the order-3 input tensors at a particular time-step, the corresponding regression target was set to be the future `PM2.5` value at the `Huairou` site. The data was then partitioned using a standard

50%-30%-20% train-validation-test split, and standardized to have zero mean and unit variance within the training set. Figure 7.6 illustrates the proposed experimental setting.

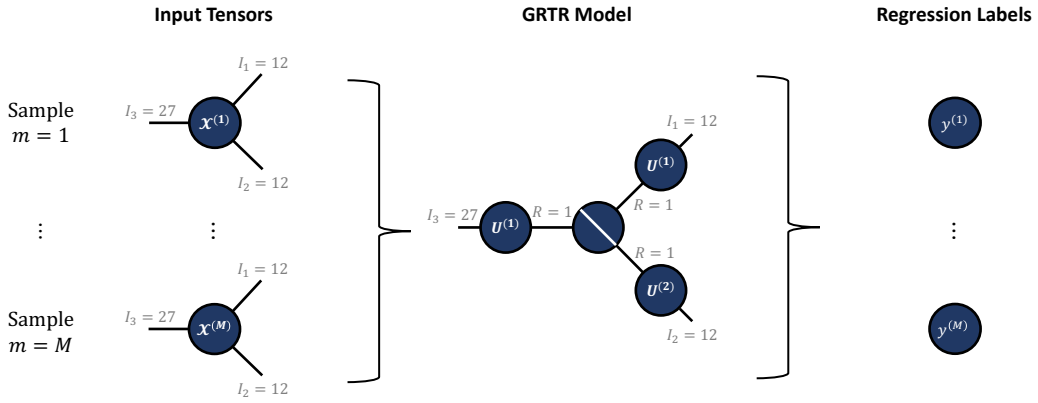


Figure 7.6: Experimental setting of the real data experiment in TN notation. The training set consists of $M = 1460$ samples of order-3 input tensors, $\boldsymbol{\mathcal{X}}^{(m)} \in \mathbb{R}^{12\times12\times27}$, and the corresponding regression labels, $y^{(m)}$. The goal of the experiment is to estimate a GRTR model from the training set data that performs well out-of-sample.

7.5.2 *Incorporation of Domain Knowledge with Graphs*

The performance of the proposed GRTR model will vary significantly depending on the design of the graph Laplacian matrix. Designing the optimal graph for air quality forecasting is beyond the scope of the chapter and is left to the domain experts. For illustrative purposes, this experiment uses a simple physical proximity graph between the various measurement sites. More specifically, using the geographical information from mode-2 (i.e., the site mode), a graph with S vertices representing S different sites was constructed, whereby the geographical distance between a pair of sites determines the edge weight through a Gaussian kernel. The corresponding Laplacian matrix for mode-2, $\boldsymbol{L}^{(2)} \in \mathbb{R}^{S\times S}$, was then computed as discussed in Section 2.2. The proposed graph Laplacian matrix encodes knowledge about the physical proximity of the mode-2 sites, which is expected to improve the learning process, as it guides the learning of the mode-2 factor matrix to be consistent with the geographical structure exhibited by the data sources.

7.5.3 *Models*

As in Section 7.4, the proposed GRTR was evaluated against: (i) LR, (ii) L2LR, (iii) standard TR, and (iv) TR with L2 regularization (L2TR). For the LR and L2LR models, the ground-truth weight tensor was vectorized as $\boldsymbol{w} = \text{vec}\,(\boldsymbol{\mathcal{W}}) \in \mathbb{R}^{12\times12\times27=3,888}$.

7.5.4 *Results*

The results for the proposed experiment are reported in Table 7.2, which validates the proposed GRTR framework, as it obtained the best scores both in terms of computational complexity costs and out-of-sample MSE.

Model	LR	L2LR	TR	L2TR	GRTR
MSE	0.741	0.134	0.077	0.084	**0.069**
Params	3,889	3,889	**52**	**52**	**52**

Table 7.2: Experimental results for the considered models. The first row shows the test set MSE achieved by the considered models. The second row shows the required number of parameters (i.e., space complexity). The proposed GRTR model achieved the best results while requiring only 52 parameters, which is comparable to the TR and the L2TR models. Linear models once again over-fitted the training set due to the large-dimensional nature of the problem.

7.6 Interpretability

By virtue of multi-linear algebra, the GRTR model offers enhanced interpretability, both in terms of individual regression coefficients and different regression dimensions.

7.6.1 *Coefficient-Wise Interpretability*

Similar to the classical LR paradigm, the proposed GRTR model allows us to recover the exact coefficient responsible for modelling a given input variable. For instance, the input value, $\boldsymbol{\mathcal{X}}_{i_1,i_2,\ldots,i_N}$, is directly linked to the coefficient, $\boldsymbol{\mathcal{W}}_{i_1,i_2,\ldots,i_N}$, of the weight tensor, which can be reconstructed from its factor matrices according to (2.9).

Example 5 *Consider the order-3 input tensor from the experiment in Section 7.5,* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{T \times S \times F}$. *The effect of the input variable,* $\boldsymbol{\mathcal{X}}_{1,10,2}$ *(i.e., the* $f = 2$*-nd feature of the* $s = 10$*-th site at the* $t = 1$*-st past time-step), on the output, is directly linked to the corresponding entry in the weight tensor,* $\boldsymbol{\mathcal{W}}_{1,10,2}$.

7.6.2 *Mode-Wise Interpretability*

By construction, the large weight tensor, $\boldsymbol{\mathcal{W}}$, is decomposed by mode-wise factor matrices, $\boldsymbol{U}^{(n)}$, which effectively breaks-down the large multi-dimensional regression problem into a set of smaller, physically meaningful, mode-wise sub-problems. This allows us to interpret the obtained model coefficients from a mode-wise point of view.

To break down the TR into a series of interpretable mode-wise operations, the TR equation in (7.1) can be rewritten as a sum of tensor contractions by applying the

rank-1 tensor inner product property discussed in Section 2.1, as

$$\begin{aligned} y &= \langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{W}} \rangle + b \\ &= \left\langle \boldsymbol{\mathcal{X}}, \sum_{r=1}^{R} \boldsymbol{u}_r^{(1)} \circ \cdots \circ \boldsymbol{u}_r^{(N)} \right\rangle + b \\ &= \sum_{r=1}^{R} \left\langle \boldsymbol{\mathcal{X}}, \boldsymbol{u}_r^{(1)} \circ \cdots \circ \boldsymbol{u}_r^{(N)} \right\rangle + b \\ &= \sum_{r=1}^{R} \boldsymbol{\mathcal{X}} \times_1^1 \boldsymbol{u}_r^{(1)} \times_2^1 \cdots \times_N^1 \boldsymbol{u}_r^{(N)} + b. \end{aligned} \tag{7.15}$$

Note that the computation of the tensor-by-vector contraction, $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{X}} \times_n^1 \boldsymbol{u}_r^{(n)}$, can be interpreted as performing a simple linear combination in the n-th subspace of $\boldsymbol{\mathcal{X}}$, as discussed in Section 2.1. Therefore, the sequence of contractions in equation (7.15), $\boldsymbol{\mathcal{X}} \times_1^1 \boldsymbol{u}_r^{(1)} \times_2^1 \cdots \times_N^1 \boldsymbol{u}_r^{(N)}$, can be interpreted as performing a sequence of linear combinations over every n-th subspace of $\boldsymbol{\mathcal{X}}$.

Example 6 *Consider the order-3 data tensor from the experiment in Section 7.5, $\boldsymbol{\mathcal{X}}$. For $R = 1$, equation (7.15) simplifies to a single sequence of contractions, $y = \boldsymbol{\mathcal{X}} \times_1^1 \boldsymbol{u}_1^{(1)} \times_2^1 \boldsymbol{u}_1^{(2)} \times_3^1 \boldsymbol{u}_1^{(3)} + b$, which can be visualized as in Figure 7.7. Observe that, TR performs effectively three linear combinations, one in each mode. The time weights, $\boldsymbol{u}_r^{(1)} \in \mathbb{R}^T$, the site weights, $\boldsymbol{u}_r^{(2)} \in \mathbb{R}^S$, and the feature weights, $\boldsymbol{u}_r^{(3)} \in \mathbb{R}^F$, determine respectively the linear weighted contribution of all T time-steps, S sites, and F features to the final regression output. For illustration, Figure 7.8 shows the estimated weights of the GRTR model with rank $R = 1$. The estimated weights can also be plotted on the underlying graph, which enhances interpretability through the visualization of the underlying relational structure, as illustrated in Figure 7.9.*

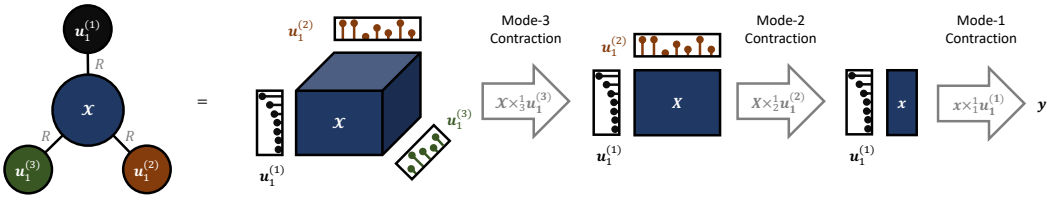


Figure 7.7: Illustration of a rank $R = 1$ order $N = 3$ TR operation. The data tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{T \times S \times F}$, contracts with the factor vectors, $\boldsymbol{u}_1^{(n)}$, $n = 1, 2, 3$, to generate the overall scalar output, $y = \boldsymbol{\mathcal{X}} \times_1^1 \boldsymbol{u}_1^{(1)} \times_2^1 \boldsymbol{u}_1^{(2)} \times_3^1 \boldsymbol{u}_1^{(3)}$. Each mode-$n$ contraction represents a linear combination in the n-th subspace of $\boldsymbol{\mathcal{X}}$, which reduces the order of the tensor until a scalar is obtained. In the above example, the original order-3 tensor is reduced to a matrix through a tensor-by-vector contraction (defined in Section 2.1), which is then reduced to a vector through a matrix-by-vector multiplication, which is then reduced to a scalar through a vector-by-vector inner product.

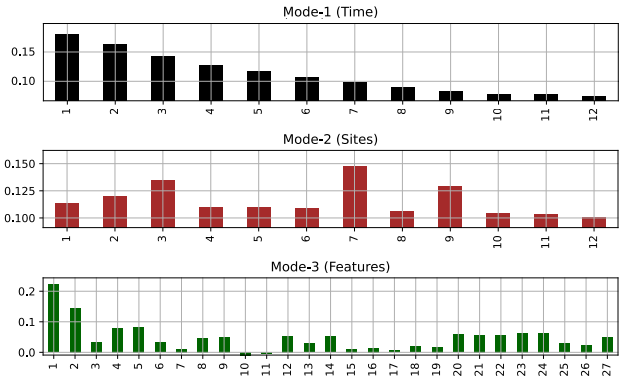


Figure 7.8: Estimated factor vectors of the rank $R = 1$ GRTR model from the experiment in Section 7.5. A factor vector is estimated for each of the time, site, and feature modes, in order to predict the future `PM2.5` values at the `Huairou` site. The time-mode factor vector, $\boldsymbol{u}^{(1)} \in \mathbb{R}^{T}$, can be interpreted as an exponentially weighted moving average operation in the time-subspace, as larger weights are placed towards more recent data. The site-mode factor vector, $\boldsymbol{u}^{(2)} \in \mathbb{R}^{S}$, indicates that the 7-th site (`Huairou`) is the most useful for forecasting its own future `PM2.5` value. The feature-mode factor vector, $\boldsymbol{u}^{(3)} \in \mathbb{R}^{F}$, indicates that the 1-st and 2-nd features (`PM2.5` and `PM10`) are the most useful for forecasting `PM2.5` value.

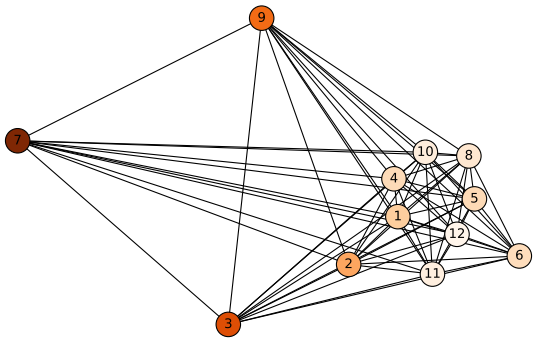


Figure 7.9: Estimated mode-2 factor vector, $\boldsymbol{u}^{(2)} \in \mathbb{R}^{S}$, of the rank $R = 1$ GRTR model from the experiment in Section 7.5, illustrated as a graph signal. The graph formulation of the proposed model allows for enhanced interpretability, as the weights of the obtained model can be plotted against the corresponding graph to reveal the effect of the underlying relational structure.

7.7 Summary

We have introduced a TR framework with graph-based regularization, which makes it possible to efficiently incorporate domain knowledge into large-scale, multi-way regression through graph Laplacian regularization. The GRTR model has been shown to be both physically meaningful and to provide enhanced results against standard vector and TR models, both in terms of out-of-sample performance and computational costs. By virtue of tensor algebra, the proposed model has also been shown to enable enhanced interpretability and control over the learning process. The proposed GRTR framework is general and opens avenues for exploring numerous possibilities for formulating physically meaningful models that accounts for intral-modal relations in big data applications.

Chapter 8

Graph Convolution via Convolution Tensors

The performance of tensor models can be greatly enhanced through graph-theoretic concepts, as discussed in Chapter 7. Similarly, this chapter discusses the use of tensor-theoretic concepts to improve graph convolutions. Convolution is the linchpin operation of modern deep learning. However, traditional convolution does not generalize to graphs due to the irregular structure of node neighbourhoods. To this end, we introduce the GCTN, which generalizes the notion of convolution tensor to general graphs. The proposed framework offers several advantages compared to existing methods in terms of efficiency, scalability, robustness, flexibility, and performance. In addition, the GCTN is general enough to include the classical CNN as a special case of convolution over circulant graphs. The benefits of the proposed framework are demonstrated through experiments, achieving state-of-the-art performance on the protein classification dataset.

8.1 Introduction

CNNs [131] are the powerhouse of modern deep learning. CNNs have achieved state-of-the-art performance in many grid-structured data applications, such as image, video, speech, and language processing tasks. However, increasingly more data are being generated by irregular sources that naturally admit a graph representation, which violates the assumptions of traditional CNNs [33]. For instance, the classical convolution operation considers a fixed number of neighbouring nodes (fixed size receptive field) that are uniquely and deterministically ranked, while for general graphs, the size of the neighbourhood varies from node to node and the relative ranking of neighbours can be ambiguous [132].

To address these issues, we propose a tensor-algebraic approach for convolution on graphs. This is done by generalizing the notion of convolution tensor to handle irregular neighbourhoods. The considered framework is shown to achieve the same complexity as classical CNNs. In addition, we show that the classical CNN can be recovered as a special case of convolution over a circulant graph. By virtue of tensor algebra, the proposed method offers several advantages: (i) it is highly

efficient, as it leverages on the sparse nature of the convolution tensor to accelerate computation; (ii) it is scalable to large graphs, as the computation can be done through batches of nodes in parallel instead of feeding in the whole graph; (iii) it is robust to perturbations in graph topology, as the learned convolution kernel is computed locally and can be adapted to changes in a node neighbourhood; (iv) it is flexible, as it can handle undirected, directed, unweighted, weighted, unsigned, and signed graphs; (v) it does not incur over-smoothing effects.

The rest of the chapter is organized as follows. Related work are first discussed in Section 8.2. The mathematical preliminaries needed to follow this work are presented in Section 8.3. The notion of convolution tensor is then discussed and generalized to graphs in Section 8.4. The proposed method is then compared against other graph convolution methods through extensive experiments in Section 8.5. The main findings of this chapter are summarized in Section 8.6.

8.2 Related Work

Several work have attempted to generalize the convolution operation from grid-structured data to general graphs. We refer readers to [37] for an overview of existing methods. Broadly, graph convolution can be either: (i) spectral, which is based on the Eigen-decomposition of the graph Laplacian matrix to perform convolution in the graph Fourier domain, or (ii) spatial, which is based on the propagation of information from neighbouring nodes to the central node similarly to how the classical convolution aggregates information from a patch of neighbouring pixels. Our work falls in the second category, which is more efficient, scalable, generalizable, and flexible compared to spectral techniques [37].

Amongst spatial graph convolution methods, GCNs [58], aggregates neighbouring node signals for each node through a weighted sum and applies a linear transform. However, this convolution-like operation is fundamentally different from the classical convolution, since: (i) the weighted sum of neighbouring signals is not trainable and is the same for all neighbouring nodes, and (ii) the receptive field varies and is determined by the neighbourhood size of each node. On the other hand, GraphSage [133] achieves fixed receptive field for every node through sampling. However, it still assumes that the contribution of neighbouring nodes is identical for all nodes. Graph Attention Networks [134] instead achieves different weights for neighbouring nodes through the attention mechanism. However, it still differs from the classical convolution as the convolution weights are a function of feature correlation between nodes instead of being learned directly, in addition to incurring more computation costs. Finally, Large-scale GCN [132] and PATCHY-SAN [135] achieves both neighbour-ranking dependent weight learning and constant receptive field. However, their focus is on transforming graph data into grid structure to enable the use of classical CNNs instead of generalizing convolution to graph data, in addition to incurring more computation costs through pre-processing.

8.3 Preliminaries

8.3.1 *Convolution and Convolution Tensor*

Discrete Convolution: Given an input vector, $\boldsymbol{x} \in \mathbb{R}^I$, and a (time-reversed) convolution kernel [136], $\boldsymbol{w} \in \mathbb{R}^K$, the discrete convolution yields the output vector, $\boldsymbol{y} = \boldsymbol{x} \star \boldsymbol{w} \in \mathbb{R}^J$, such that $\boldsymbol{y}_j = \sum_{k=1}^{K} \boldsymbol{x}_{j+k-1}\boldsymbol{w}_k$ for $j = 1, \ldots, J$, where $J = I - K + 1$ and $I > K$. In practice, it is often desirable to ensure that the output and input vectors have the same size, $J = I$. This is achieved by assuming circularity in data (the last entry is connected to the first entry in a circular fashion), which modifies the convolution such that $\boldsymbol{y}_j = \sum_{k=1}^{K} \boldsymbol{x}_{(j+k-1)\%I}\boldsymbol{w}_k$, where $\%$ denotes the modulo operation.

Convolution Tensor: The classical 1D convolution tensor [19], $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{J\times I\times K}$, where $I > K$ and $J = I - K + 1$, is a sparse order-3 tensor with entries $\boldsymbol{\mathcal{T}}_{j,j+k-1,k} = 1$ for $j = 1, \ldots, J$ and $k = 1, \ldots, K$. Given an input vector, $\boldsymbol{x} \in \mathbb{R}^I$, and a convolution kernel, $\boldsymbol{w} \in \mathbb{R}^K$, the discrete convolution can be obtained through the tensor contraction $\boldsymbol{y} = \boldsymbol{x} \star \boldsymbol{w} = \boldsymbol{\mathcal{T}} \times_2^1 \boldsymbol{x} \times_3^1 \boldsymbol{w} \in \mathbb{R}^J$. If circularity is assumed, the convolution tensor entries is modified such that $\boldsymbol{\mathcal{T}}_{j,(j+k-1)\%I,k} = 1$ for $j = 1, \ldots, J = I$ and $k = 1, \ldots, K$, where $\%$ denotes the modulo operation.

8.4 Convolution Tensor for Graphs

8.4.1 *Circulant Graph Nature of the Convolution Tensor*

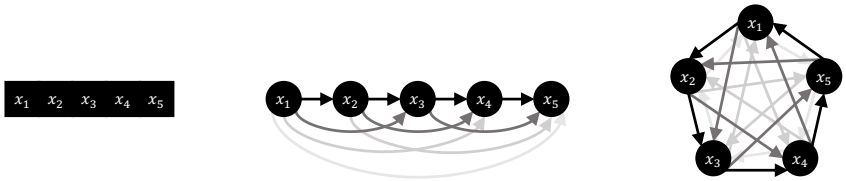


Figure 8.1: Input signal vector, $\boldsymbol{x} \in \mathbb{R}^5$, represented as a vector (left), a directed graph signal (middle), and DCG signal (right). The nodes represent time-steps, while the edges connect the time-steps in time-increasing order with weights proportional to the time difference.

Consider an input signal vector, $\boldsymbol{x}$. The input vector can be interpreted as the graph signal of a directed graph with I nodes representing I successive time-steps that are connected in time-increasing order. If circularity is assumed, then the last time-steps are connected to the first time-steps in a circulant structure. Figure 8.1 illustrates the graph representation of classical signals.

Given a convolution kernel, $\boldsymbol{w}$, the discrete convolution (with circularity assumption)

$$\boldsymbol{y}_j = \sum_{k=1}^{K} \boldsymbol{x}_{(j+k-1)\%I}\boldsymbol{w}_k \tag{8.1}$$

can be interpreted to compute for every node, j, a weighted sum of graph signals over its K nearest neighbour nodes, $i \in \mathcal{N}_j$, where the k-th nearest neighbour of the j-th node is the $(j+k-1)$-th node, $k = 1, \ldots, K$. This is reflected in the structure of the convolution tensor as defined in Section 8.3.1, which is a multi-linear map that links every node, j, to its K nearest neighbour nodes, $i \in \mathcal{N}_j$, and the corresponding convolution kernel weights in $\boldsymbol{w}$. The relationship between the discrete convolution operation and the convolution tensor is illustrated in Figure 8.2 and Figure 8.3.

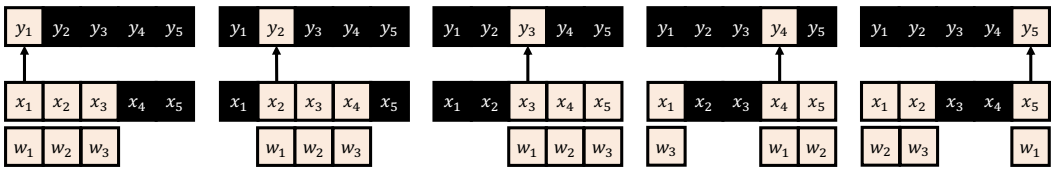


Figure 8.2: Standard representation of the discrete convolution. This can be written in element-wise form as $\boldsymbol{y}_j = \sum_{k=1}^{K} \boldsymbol{x}_{(j+k-1)\%I} \boldsymbol{w}_k$. Note that the indices of the element-wise variables, $(\boldsymbol{y}_j, \boldsymbol{x}_{(j+k-1)\%I}, \boldsymbol{w}_k)$, correspond exactly to the non-zero entries of the convolution tensor, $\boldsymbol{\mathcal{T}}_{j,j+k-1,k} = 1$ for $j = 1, \ldots, J$ and $k = 1, \ldots, K$.

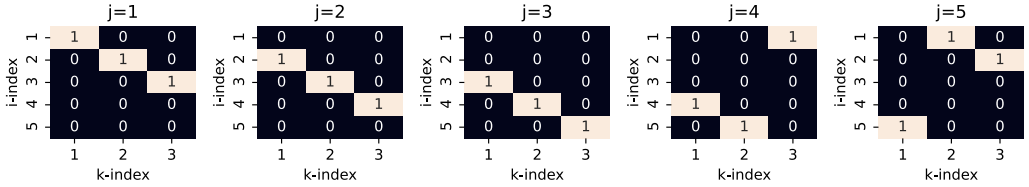


Figure 8.3: Slices of the convolution tensor, $\boldsymbol{\mathcal{T}}_{j,:,:} \in \mathbb{R}^{I \times K}$ for $j = 1, \ldots, J$, where $I = J = 5$ and $K = 3$. The non-zero entries of the convolution tensor links each output entry (the j-th node), to its K nearest neighbour input entries (the neighbouring nodes, $i \in \mathcal{N}_j$), and the corresponding convolution kernel entries. For instance, for the $j = 1$ output entry, $\boldsymbol{y}_1$, its $K = 3$ nearest neighbour graphs signals are associated with the $i = 1, 2, 3$ input entries, $(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$, which are mapped to the $k = 1, 2, 3$ kernel entries, $(\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3)$. Instead, for the $j = 5$ output entry, $\boldsymbol{y}_5$, its $K = 3$ nearest neighbour graph signals are associated with the $i = 5, 1, 2$ input entries, $(\boldsymbol{x}_5, \boldsymbol{x}_1, \boldsymbol{x}_2)$, which are mapped to the $k = 1, 2, 3$ kernel entries, $(\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3)$.

The convolution operation in (8.1) can be written in the well-known circulant matrix form, $\boldsymbol{y} = \tilde{\boldsymbol{W}}\boldsymbol{x}$, where $\tilde{\boldsymbol{W}}_{j,(j+k-1)\%I} = \boldsymbol{w}_k$ for $j = 1, \ldots, J$ and $k = 1, \ldots, K$. Under the GSP framework, the circulant matrix, $\tilde{\boldsymbol{W}} \in \mathbb{R}^{J \times I}$, is effectively the adjacency matrix of a DCG, as defined in Section 2.2. Therefore, the classical convolution can be understood as a graph operation over a DCG, where the entries of the associated adjacency matrix, $\tilde{\boldsymbol{W}}$, is parameterized by the convolution kernel, $\boldsymbol{w}$, as a function of K nearest neighbours.

The convolution tensor effectively acts as a generator of the adjacency matrix, $\tilde{\boldsymbol{W}}$. Indeed, using the properties discussed in section 8.3.1, the tensor contraction based convolution defined in Section 8.3.1 can be re-written in the circulant matrix

form as

$$\boldsymbol{y} = \boldsymbol{\mathcal{T}} \times_2^1 \boldsymbol{x} \times_3^1 \boldsymbol{w} = \boldsymbol{\mathcal{T}} \times_3^1 \boldsymbol{w} \times_2^1 \boldsymbol{x} = \left(\boldsymbol{\mathcal{T}} \times_3^1 \boldsymbol{w}\right) \boldsymbol{x} = \left(\sum_{k=1}^{K} \boldsymbol{w}_k \boldsymbol{\mathcal{T}}_{:,:,k}\right) \boldsymbol{x} = \tilde{\boldsymbol{W}} \boldsymbol{x} \quad (8.2)$$

where $\boldsymbol{w}_k$ is the k-th entry of the convolution kernel and $\boldsymbol{\mathcal{T}}_{:,:,k} \in \mathbb{R}^{I \times I}$ is the k-th slice of the convolution tensor, $k = 1, \ldots, K$. The adjacency matrix is therefore decomposed as a weighted sum of K convolution tensor slices as

$$\tilde{\boldsymbol{W}} = \sum_{k=1}^{K} \boldsymbol{w}_k \boldsymbol{\mathcal{T}}_{:,:,k} \quad (8.3)$$

where the k-th slice effectively models the k-th nearest neighbour connections of the DCG, as illustrated in Figure 8.4.

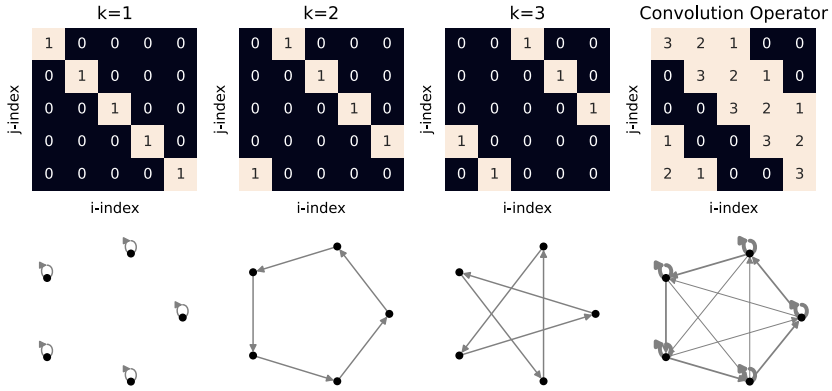


Figure 8.4: Illustration of the adjacency matrix, $\tilde{\boldsymbol{W}}$, generated using the convolution tensor, $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{5 \times 5 \times 3}$, and the convolution kernel, $\boldsymbol{w} = [3, 2, 1] \in \mathbb{R}^3$, according to (8.3). The top figures illustrate: (i) the $k = 1$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,1} \in \mathbb{R}^{5 \times 5}$; (ii) the $k = 2$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,2} \in \mathbb{R}^{5 \times 5}$; (iii) the $k = 3$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,3} \in \mathbb{R}^{5 \times 5}$; (iv) the adjacency matrix computed as a weighted sum of convolution tensor slices, $\tilde{\boldsymbol{W}} = \sum_{k=1}^{3} \boldsymbol{w}_k \boldsymbol{\mathcal{T}}_{:,:,k} \in \mathbb{R}^{5 \times 5}$. The bottom figures illustrate the corresponding graph topology implied by the adjacency matrix in the top figures. Note that the k-th tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,k}$, models the k-th nearest neighbour connection. For instance, $k = 1$ models self-connections, $k = 2$ models connections to successive time-steps, and $k = 3$ models connections 2 time-steps away.

Remark 15 *The generation of the adjacency matrix in (8.3) captures the parameter sharing property of convolution, as the k-th convolution kernel parameter, w_k, is shared with the k-th nearest neighbour of all nodes.*

Remark 16 *The adjacency matrix, $\tilde{\boldsymbol{W}} \in \mathbb{R}^{J \times I}$, effectively acts as a convolution operator with respect to a DCG.*

8.4.2 *Convolution Tensor for 2D Convolution*

The 1D convolution tensor can be extended to 2D by doubling the dimensions. More specifically, the 2D convolution tensor, $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{J_1 \times J_2 \times I_1 \times I_2 \times K_1 \times K_2}$, where $I_1 > K_1$, $I_2 > K_2$, $J_1 = I_1 - K_1 + 1$ and $J_2 = I_2 - K_2 + 1$, is a sparse order-6 tensor with entries $\boldsymbol{\mathcal{T}}_{j_1,j_2,j_1+k_1-1,j_2+k_2-1,k_1,k_2} = 1$ for $j_1 = 1,\ldots,J_1$, $j_2 = 1,\ldots,J_2$, $k_1 = 1,\ldots,K_1$, and $k_2 = 1,\ldots,K_2$. Given an input matrix, $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$, and a convolution kernel, $\boldsymbol{W} \in \mathbb{R}^{K_1 \times K_2}$, the 2D discrete convolution can be obtained through the tensor contraction $\boldsymbol{Y} = \boldsymbol{X} \star \boldsymbol{W} = \boldsymbol{\mathcal{T}} \times_{3,4}^{1,2} \boldsymbol{X} \times_{5,6}^{1,2} \boldsymbol{W} \in \mathbb{R}^{J_1 \times J_2}$. If circularity is assumed, the convolution tensor entries is modified such that $\boldsymbol{\mathcal{T}}_{j_1,j_2,(j_1+k_1-1)\%I_1,(j_2+k_2-1)\%I_2,k_1,k_2} = 1$ for $j_1 = 1,\ldots,J_1$, $j_2 = 1,\ldots,J_2$, $k_1 = 1,\ldots,K_1$, and $k_2 = 1,\ldots,K_2$, where $\%$ denotes the modulo operation. The associated 2D convolution operator, $\tilde{\boldsymbol{\mathcal{W}}} = \left(\boldsymbol{\mathcal{T}} \times_{5,6}^{1,2} \boldsymbol{W}\right) \in \mathbb{R}^{J_1 \times J_2 \times I_1 \times I_2}$, is an order-4 tensor, as illustrated in Figure 8.6.

For simplicity, the $2D$ convolution tensor can be reshaped as an order-3 tensor, $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{J \times I \times K}$, where $J = J_1 J_2$, $I = I_1 I_2$, and $K = K_1 K_2$. The $2D$ discrete convolution can then be computed as $\boldsymbol{y} = \boldsymbol{\mathcal{T}} \times_2^1 \boldsymbol{x} \times_3^1 \boldsymbol{w} \in \mathbb{R}^J$, where $\boldsymbol{y} \in \mathbb{R}^J$, $\boldsymbol{x} \in \mathbb{R}^I$, and $\boldsymbol{w} \in \mathbb{R}^K$ are respectively the vectorized forms of matrices $\boldsymbol{Y} \in \mathbb{R}^{J_1 \times J_2}$, $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$, and $\boldsymbol{W} \in \mathbb{R}^{K_1 \times K_2}$. In this way, the associated convolution operator reduces to a doubly block-circulant matrix, $\tilde{\boldsymbol{W}} \in \mathbb{R}^{J \times I}$, as illustrated in Figure 8.5.

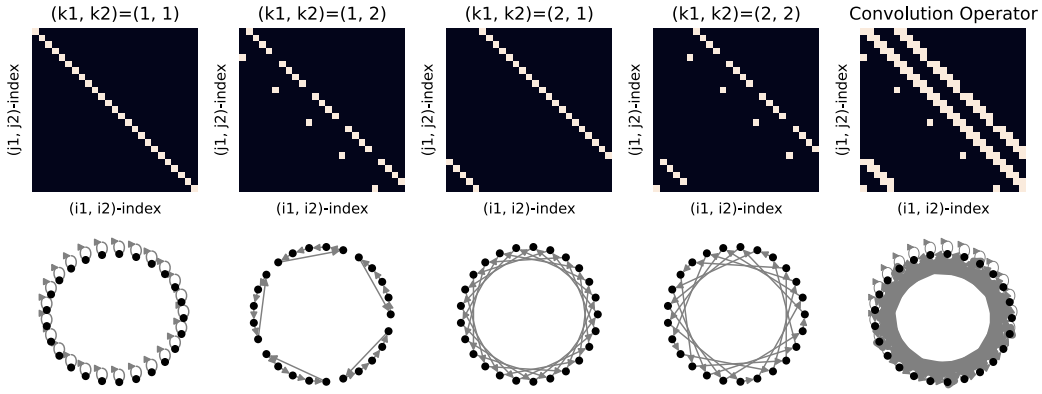


Figure 8.5: Illustration of the convolution operator, $\tilde{\boldsymbol{W}} \in \mathbb{R}^{J \times I}$, generated using the convolution tensor, $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{J \times I \times K}$, and the convolution kernel, $\boldsymbol{w} = [1,2,3,4] \in \mathbb{R}^K$. The top figures illustrate: (i) the $(k_1,k_2) = (1,1)$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,1} \in \mathbb{R}^{J \times I}$; (ii) the $(k_1,k_2) = (1,2)$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,2} \in \mathbb{R}^{J \times I}$; (iii) the $(k_1,k_2) = (2,1)$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,3} \in \mathbb{R}^{J \times I}$; (iv) the $(k_1,k_2) = (2,2)$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,4} \in \mathbb{R}^{J \times I}$; (v) the convolution operator computed as a weighted sum of convolution tensor slices, $\tilde{\boldsymbol{W}} = \sum_{k=1}^{K} \boldsymbol{w}_k \boldsymbol{\mathcal{T}}_{:,:,k} \in \mathbb{R}^{J \times I}$. The bottom figures illustrate the corresponding graph topology implied by the adjacency matrix in the top figures. Note that the k-th tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,k}$, models the k-th nearest neighbour connection.

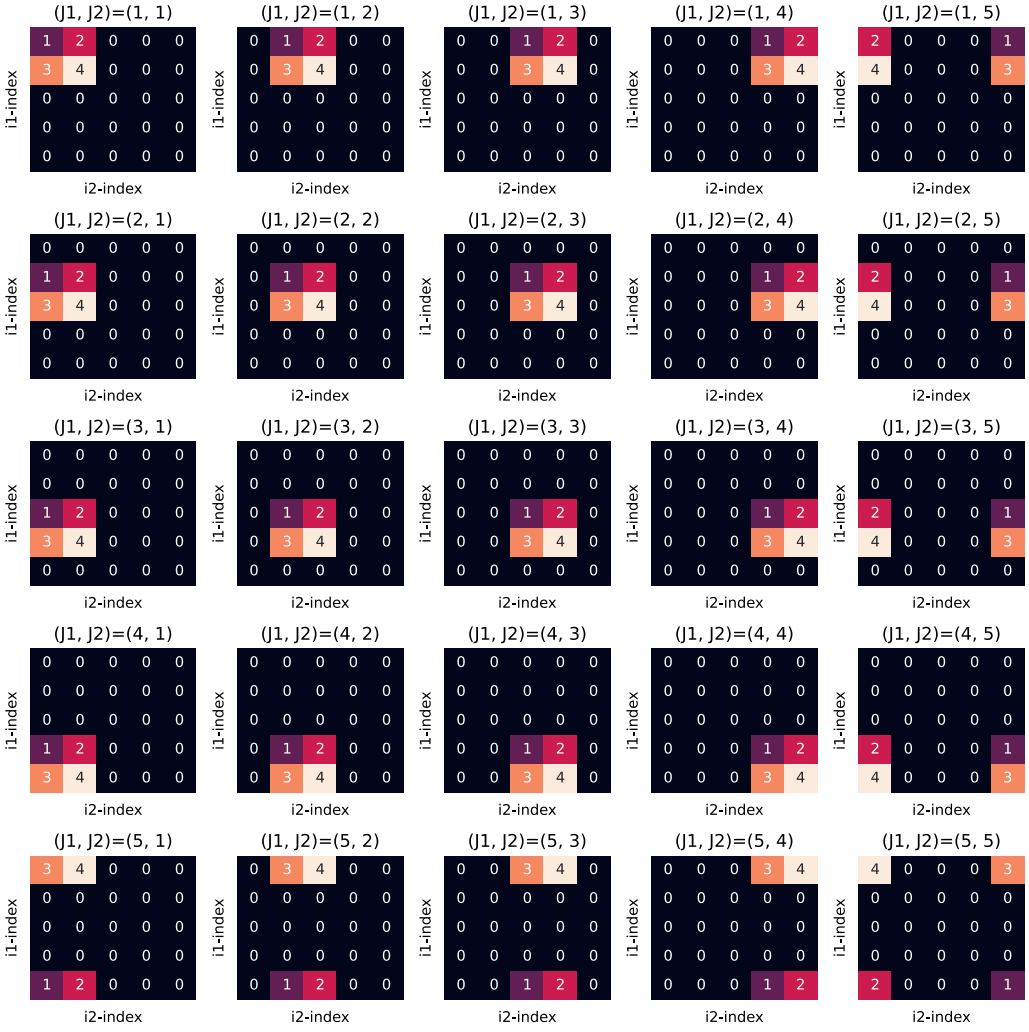


Figure 8.6: Illustration of the 2D convolution operator, $\tilde{\boldsymbol{\mathcal{W}}} = \left(\boldsymbol{\mathcal{T}} \times_{5,6}^{1,2} \boldsymbol{W}\right) \in \mathbb{R}^{J_1 \times J_2 \times I_1 \times I_2}$, given the convolution kernel, $\boldsymbol{W} = [[1, 2], [3, 4]] \in \mathbb{R}^{K_1 \times K_2}$, where $I_1 = I_2 = J_1 = J_2 = 5$ and $K_1 = K_2 = 2$. The (j_1, j_2) sub-figure illustrates the corresponding convolution operator slice, $\tilde{\boldsymbol{\mathcal{W}}}_{j_1, j_2, :, :} \in \mathbb{R}^{I_1 \times I_2}$.

8.4.3 *General Graph Convolution Tensors*

The discrete convolution (and the associated convolution tensor) assumes a circulant graph with highly regular topology, which is reflected in the circulant nature of the adjacency matrix, $\tilde{\boldsymbol{W}}$. This regular structure is appropriate for processing classical signals sampled at regular and uniform intervals (grid-structured data), such as speech and images, but is not directly applicable to general graphs with irregular structures. To extend the notion of convolution tensor to general graphs with irregular structures, there are three core challenges that must be addressed:

1. **Number of neighbouring nodes.** In classical convolution, the size of the convolution kernel, K, determines number of the neighbouring nodes to be processed (the receptive field), which is kept constant for all nodes by virtue of its grid-like structure. However, for general graphs, the number of the neighbouring nodes can be limited and varies for each node. To this end, let K be a hyper-parameter. For nodes where $|\mathcal{N}_j| > K$, the graph convolution can be performed with the first K neighbours only. For nodes where $|\mathcal{N}_j| < K$, the convolution kernel can be masked such that $\boldsymbol{w}_k = 0$ for $k > |\mathcal{N}_j|$. The largest possible choice of K for general graphs is bounded by the size of the largest neighbourhood in the graph, $K = \max_j\{|\mathcal{N}_j|\}$.

2. **Ranking of neighbouring nodes.** In classical convolution, the ranking of the K nearest neighbours is trivial, as discussed in Section 8.4.1. For weighted graphs, the ranking of neighbouring nodes can be determined by the respective edge weights.

3. **Proximity of neighbouring nodes.** In classical convolution, the proximity between a node, j, and its k-th closest neighbour is constant for all j. However, for general graphs, the proximity between a node, j, and its k-th nearest neighbour varies depending on the edge weight for every j. To this end, the entries of the convolution tensor should be modulated by the corresponding edge weights to reflect the proximity information.

Algorithm 5 extends the convolution tensor to general graphs by applying the above considerations. Formally, given the adjacency matrix of an irregular graph, $\boldsymbol{A} \in \mathbb{R}^{J\times I}$, and the receptive field hyperparameter (convolution kernel size), K, Algorithm 5 generates a graph convolution tensor, $\boldsymbol{\mathcal{T}} = f(\boldsymbol{A}, K)$, such that $\boldsymbol{y} = \boldsymbol{\mathcal{T}} \times_2^1 \boldsymbol{x} \times_3^1 \boldsymbol{w} \in \mathbb{R}^J$ performs convolution over the irregular graph domain.

Remark 17 *For the special case of DCGs, Algorithm 5 yields the classical convolution tensor as defined in Section 8.3.1, which implements the classical convolution.*

Remark 18 *The convolution tensor entry, $\boldsymbol{\mathcal{T}}_{j,i,k}$, can be modulated by any kind of graph edge information, $\boldsymbol{A}_{j,i}$. This allows the convolution operation to be defined over directed, undirected, weighted, unweighted, signed, and unsigned graphs.*

Remark 19 *Perturbations to the graph topology will change the underlying convolution tensor, $\boldsymbol{\mathcal{T}}$, as the K nearest neighbour ranking information must be updated accordingly.*

However, this does not affect in any way the learned convolution kernel, $\boldsymbol{k} \in \mathbb{R}^K$, which can be applied to the new graph topology without re-training, which makes it robust to changes in the graph.

Algorithm 5: Graph Convolution Tensor

Input: Weighted adjacency matrix, $\boldsymbol{A} \in \mathbb{R}^{J \times I}$, convolution kernel size, K.
Output: Graph Convolution Tensor, $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{J \times I \times K}$
Initialise $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{J \times I \times K}$ with zeros.
for $j = 1, \ldots, J$ **do** `// for each of the` j`-th node`
  for $i = 1, \ldots, I$ **do** `// for every other` i`-th node`
    if $\boldsymbol{A}_{j,i} \neq 0$ **then** `// if` $i \in \mathcal{N}_j$
      k = `ranking`$(\boldsymbol{A}, j, i)$ `// ranking of the` i`-th node within` $\mathcal{N}_j$
      if $k <= K$ **then** `// consider the first` K `nearest neighbours`
        $\boldsymbol{\mathcal{T}}_{j,i,k} = \boldsymbol{A}_{j,i}$ `// modulate convolution tensor entry by edge weight`
      end
    end
  end
end
Return: $\boldsymbol{\mathcal{T}}$

Given a graph convolution tensor, $\boldsymbol{\mathcal{T}}$, an implied adjacency matrix can be computed as $\tilde{\boldsymbol{W}} = \boldsymbol{\mathcal{T}} \times_3^1 \boldsymbol{w} = \sum_{k=1}^K \boldsymbol{w}_k \boldsymbol{\mathcal{T}}_{:,:,k}$, which effectively acts as the convolution operator over the considered graph domain. The non-zero entries (edges) of the convolution operator, $\tilde{\boldsymbol{W}}_{j,i}$, are determined by the non-zero entries of the original adjacency matrix, $\boldsymbol{A}_{j,i}$, while the values of $\tilde{\boldsymbol{W}}_{j,i}$ are parameterized by the convolution kernel, $\boldsymbol{w}$, as a function of K nearest neighbours. This is illustrated intuitively in Example 7.

Remark 20 *If the hyper-paramter, K, is chosen to be the size of the largest neighbourhood in the graph, $K = \max_j\{|\mathcal{N}_j|\}$, then the graphs implied by $\tilde{\boldsymbol{W}}$ and $\boldsymbol{A}$ are topologically equivalent (same set of edges but with different edge weights). If $K < \max_j\{|\mathcal{N}_j|\}$, then the graph implied by $\tilde{\boldsymbol{W}}$ only contains a subset of the edges contained in the original graph implied by $\boldsymbol{A}$ (only the first K nearest neighbour edges of every node are kept).*

Example 7 *Consider an irregular graph with the following adjacency matrix ($I = J = 5$)*

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0.1 & 0 & 0.01 & 0 \\ 0.01 & 1 & 0 & 0 & 0.1 \\ 0.1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0.1 & 0 & 0 & 1 \end{pmatrix} \tag{8.4}$$

Given the convolution kernel, $\boldsymbol{w} = [3, 2, 1] \in \mathbb{R}^3$ ($K = 3$), the slices of the graph convolution tensor, $\boldsymbol{\mathcal{T}}_{:,:,k} \in \mathbb{R}^{5 \times 5}$, and the associated convolution operator, $\tilde{\boldsymbol{W}} = \boldsymbol{\mathcal{T}} \times_3^1 \boldsymbol{w} = \sum_{k=1}^K \boldsymbol{w}_k \boldsymbol{\mathcal{T}}_{:,:,k} \in \mathbb{R}^{5 \times 5}$, are illustrated in Figure 8.7.

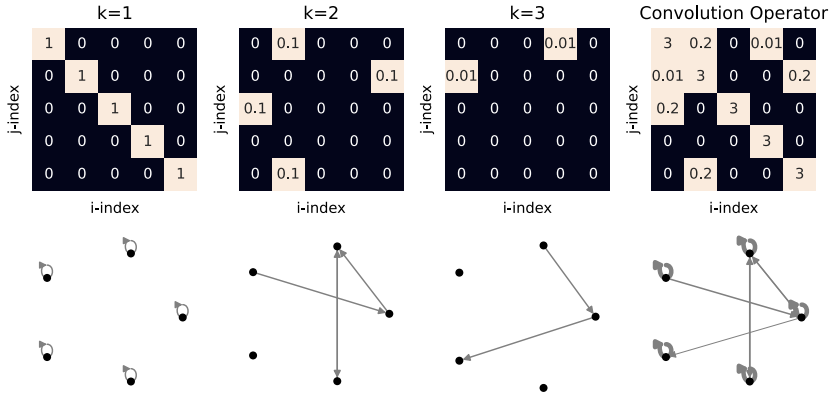


Figure 8.7: Illustration of the convolution operator, $\tilde{\boldsymbol{W}} \in \mathbb{R}^{5\times5}$, generated using the convolution tensor, $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{5\times5\times3}$, and the convolution kernel, $\boldsymbol{w} = [3, 2, 1] \in \mathbb{R}^3$, associated with Example 7. The top figures illustrate: (i) the $k = 1$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,1} \in \mathbb{R}^{5\times5}$; (ii) the $k = 2$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,2} \in \mathbb{R}^{5\times5}$; (iii) the $k = 3$ convolution tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,3} \in \mathbb{R}^{5\times5}$; (iv) the convolution operator computed as a weighted sum of convolution tensor slices, $\tilde{\boldsymbol{W}} = \sum_{k=1}^{3} \boldsymbol{w}_k \boldsymbol{\mathcal{T}}_{:,:,k} \in \mathbb{R}^{5\times5}$. The bottom figures illustrate the corresponding graph topology implied by the adjacency matrix in the top figures. Note that the k-th tensor slice, $\boldsymbol{\mathcal{T}}_{:,:,k}$, models the k-th nearest neighbour connection. In contrast to the classical convolution tensor illustrated in Figure 8.4, the irregular structure of the graph is reflected in the irregular positions of non-zero entries in the convolution tensor slices, since not every node has a k-th nearest neighbour and the ranking of neighbouring nodes is not successive.

8.4.4 *Interpretation of Graph Convolution via Graph Convolution Tensor*

Given the adjacency matrix of a graph, $\boldsymbol{A}$, and the convolution kernel size hyperparameter, K, the graph convolution operation can be computed as $\boldsymbol{y} = \boldsymbol{\mathcal{T}} \times_2^1 \boldsymbol{x} \times_3^1 \boldsymbol{w}$, where $\boldsymbol{\mathcal{T}} = f(\boldsymbol{A}, K)$ is the graph convolution tensor generated according to Algorithm 5, $\boldsymbol{x}$ is the input graph signal, and $\boldsymbol{w}$ is the convolution kernel. Equivalently, the graph convolution can be written in element-wise form for each node j as

$$\boldsymbol{y}_j = \sum_{i \in \mathcal{N}_j^{(K)}} \boldsymbol{w}_k \boldsymbol{A}_{j,i} \boldsymbol{x}_i \tag{8.5}$$

where $\boldsymbol{y}_j$ is the output signal at node j, $\mathcal{N}_j^{(K)}$ denotes the set of K nearest neighbours of node j, $\boldsymbol{w}_k$ is the k-th entry of the convolution kernel with k being the ranking of node i within $\mathcal{N}_j^{(K)}$, $\boldsymbol{A}_{j,i}$ is the (j, i)-th entry of the adjacency matrix (edge weight), and $\boldsymbol{x}_i$ is the input signal at node i. Therefore, (8.5) can be interpreted as the propagation of neighbouring signals along graph edges that is parameterized by the learnable convolution kernel as a function of neighbourhood node ranking.

Remark 21 *The entries of the convolution kernel can be learned directly for all K nearest neighbours and can take on any real number value. This is fundamentally different*

from other convolution methods where the convolution parameters are restricted to compute a local weighted average operation (Laplacian smoothing) which can lead to over-smoothing effects through repeated applications [137].

8.4.5 *Efficient and Scalable Graph Convolution*

In practice, the tensor contraction, $\boldsymbol{y} = \boldsymbol{\mathcal{T}} \times_2^1 \boldsymbol{x} \times_3^1 \boldsymbol{w}$, can be computed sequentially in two equivalent ways: (i) $\boldsymbol{y} = (\boldsymbol{\mathcal{T}} \times_3^1 \boldsymbol{w})\, \boldsymbol{x} = \tilde{\boldsymbol{W}} \boldsymbol{x}$, or (ii) $\boldsymbol{y} = (\boldsymbol{\mathcal{T}} \times_2^1 \boldsymbol{x})\, \boldsymbol{w} = \tilde{\boldsymbol{X}} \boldsymbol{w}$. For computational efficiency, the convolution tensor should be stored in sparse format, which enables highly efficient computation of dense matrices, $\tilde{\boldsymbol{W}} = (\boldsymbol{\mathcal{T}} \times_3^1 \boldsymbol{w})$ and $\tilde{\boldsymbol{X}} = (\boldsymbol{\mathcal{T}} \times_2^1 \boldsymbol{x})$. Therefore, the bottleneck of the overall computational complexity is determined by the dense multiplications, $\boldsymbol{y} = \tilde{\boldsymbol{W}} \boldsymbol{x}$ and $\boldsymbol{y} = \tilde{\boldsymbol{X}} \boldsymbol{w}$, which have respectively a complexity of $\mathcal{O}(JI)$ and $\mathcal{O}(JK)$. Given that the size of the convolution kernel is typically chosen to be much smaller than the number of nodes in a graph, $K << I$, the second method yield significantly more efficient computation. Figure 8.8 illustrates the structure of matrices $\tilde{\boldsymbol{W}} \in \mathbb{R}^{J \times I}$ and $\tilde{\boldsymbol{X}} \in \mathbb{R}^{J \times K}$ associated with the classical convolution.

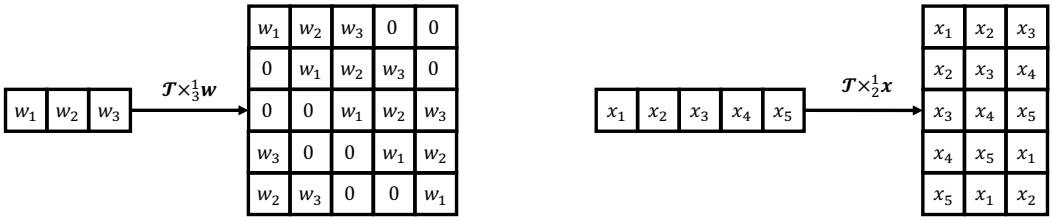


Figure 8.8: The structure of matrices $\tilde{\boldsymbol{W}} \in \mathbb{R}^{J \times K}$ and $\tilde{\boldsymbol{X}} \in \mathbb{R}^{J \times I}$ generated from $\boldsymbol{w} \in \mathbb{R}^3$ and $\boldsymbol{x} \in \mathbb{R}^5$.

The proposed method consider for every node, j, at most K nearest neighbour nodes to perform the convolution operation. Therefore, for a batch of B nodes, the convolution operation considers at most BK neighbouring nodes for computation, which is typically much smaller than the total number of nodes in a graph, $BK << I$. Given that the nearest neighbour relationship is determined prior to convolution, this allows the computation to be separated in batches of nodes and processed in parallel, which makes the considered method highly scalable to large graphs.

8.4.6 *Multi-Channel Graph Convolution*

Similar to classical CNNs, for multi-channel graph data, $\boldsymbol{X} \in \mathbb{R}^{I \times C}$, the convolution kernel, $\boldsymbol{W} \in \mathbb{R}^{K \times C}$, becomes a matrix to cover all C channels. This effectively modifies the tensor contraction based convolution to $\boldsymbol{y} = (\boldsymbol{\mathcal{T}} \times_2^1 \boldsymbol{X}) \times_{2,3}^{1,2} \boldsymbol{W} = \tilde{\boldsymbol{\mathcal{X}}} \times_{2,3}^{1,2} \boldsymbol{W} \in \mathbb{R}^J$, where $\tilde{\boldsymbol{\mathcal{X}}} = (\boldsymbol{\mathcal{T}} \times_2^1 \boldsymbol{X}) \in \mathbb{R}^{J \times K \times C}$. To generate multi-channel convolution outputs, a set of F different convolution kernels can be defined and stacked as an order-3 tensor, $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{K \times C \times F}$, which results in the multi-channel output, $\boldsymbol{Y} = (\boldsymbol{\mathcal{T}} \times_2^1 \boldsymbol{X}) \times_{2,3}^{1,2} \boldsymbol{\mathcal{W}} = \tilde{\boldsymbol{\mathcal{X}}} \times_{2,3}^{1,2} \boldsymbol{\mathcal{W}} \in \mathbb{R}^{J \times F}$.

8.4.7 *Graph Convolution Tensor Network*

Given the adjacency matrix of a graph, $\boldsymbol{A} \in \mathbb{R}^{J \times I}$, and the convolution kernel size hyper-parameter, K, we define the forward pass of the GCTN as

$$\boldsymbol{Y} = \sigma\left(\left(\boldsymbol{\mathcal{T}} \times_2^1 \boldsymbol{X}\right) \times_{2,3}^{1,2} \boldsymbol{\mathcal{W}} + \boldsymbol{b}\right) \in \mathbb{R}^{J \times F} \tag{8.6}$$

where $\boldsymbol{Y} \in \mathbb{R}^{J \times F}$ is the multi-channel graph convolution output, $\boldsymbol{\mathcal{T}} = f\left(\boldsymbol{A}, K\right) \in \mathbb{R}^{J \times I \times K}$ is the graph convolution tensor generated according to Algorithm 5, $\boldsymbol{X} \in \mathbb{R}^{I \times C}$ is the multi-channel graph signals input, $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{K \times C \times F}$ is the trainable convolution kernel, $\boldsymbol{b} \in \mathbb{R}^{F}$ is the trainable additive bias, and $\sigma(\cdot)$ is an activation function. The number of trainable parameters in the proposed GCTN scales with the same space complexity as the Classical CNN, $\mathcal{O}\left(KCF + B\right)$, which is independent of number of nodes.

8.5 Experiments

This section outlines the experimental setup and methodology for evaluating the proposed GCTN in the context of graph classification.

Dataset. The PROTEIN dataset was introduced by Karsten M. Borgwardt *et al.* in the article "Protein Function Prediction via Graph Kernels" [138]. This dataset contains proteins in a graph form that are either enzymes or not. The nodes of each graph are amino acids and an edge is formed if two nodes are less than 6 Ångströms apart. The dataset was cleaned by removing isomorphic graphs, resulting in 1113 proteins in total.

Model. The experiment model contained: (i) A GCTN layer with the ReLU activation to perform node-level aggregation; (ii) Three parallel global pooling layers (average-pooling, max-pooling, and sum-pooling) with concatenated outputs; (iii) A DNN with 32 hidden units; (iv) A final DNN with 2 hidden units acting as logits. The NAdam optimizer with a learning rate of 10^{-6} was used. An early stopping mechanism was employed such that the training was halted if the training loss did not decrease for five consecutive epochs. The implementation was performed in TensorFlow on an Intel(R) Xeon(R) CPU @ 2.30GHz.

Results. To evaluate the performance of the proposed model. The GCTN model was compared against a range of state-of-the-art baselines for the given dataset, including sGIN [139], U2GNN [140], MEWISPool [141], DUGNN [142], and HGP-SL [143]. We utilized the same 10-fold cross validation method employed in [139, 140, 141, 142] for evaluation. Table 8.1 illustrates classification accuracy achieved by the considered models, where the proposed GCTN model achieved the best score of $87.96 \pm 17.51\%$. A comparison of parameters was also conducted against the baselines as a measure of space complexity (parameters are None for models that did not offer any parameter information). Notably, in addition to achieving superior performance, the proposed GCTN also demonstrated a significant advantage in terms of its lightweight nature.

Remark 22 *By design, the same set of GCTN convolution kernels was used for different proteins. This testifies to the robustness of the GCTN model to perturbations in graph topology, where the learned kernels can be easily adapted to graphs with different structures (i.e., different proteins).*

	sGIN	U2GNN	MEWISPool	DUGNN	HGP-SL	**GCTN**
Accuracy	78.97%	80.01%	80.71%	81.70%	84.91%	**87.96%**
Parameters	None	1252	None	4192	60800	**566**

Table 8.1: Performance comparison on the PROTEIN dataset.

8.6 Summary

We have introduced a graph convolution method based on the convolution tensor, which offers several advantages over existing methods in terms of efficiency, scalability, robustness, flexibility and performance. The so proposed GCTN is also shown to be general enough to include the classical CNN as a special case of convolution over circulant graphs. The benefits of the GCTN are validated through experiments, achieving state-of-the-art performance on the protein classification dataset.

Chapter 9

Recurrent Graph Tensor Networks

This chapter analyzes RNNs through a joint consideration of tensors and graphs. RNNs are among the most successful ML models for sequence modelling, but tend to suffer from an exponential increase in the number of parameters when dealing with large multidimensional data. To this end, we develop a multi-linear graph filter framework for approximating the modelling of hidden states in RNNs, which is embedded in a TN architecture to improve modelling power and reduce parameter complexity, resulting in a novel Recurrent GTN (RGTN). The proposed framework is validated through several multi-way sequence modelling tasks and benchmarked against traditional RNNs. By virtue of the domain aware information processing of graph filters and the expressive power of TNs, we show that the proposed RGTN is capable of not only out-performing standard RNNs, but also mitigating the curse of dimensionality associated with traditional RNNs, demonstrating superior properties in terms of performance and complexity.

9.1 Introduction

Graphs and tensors have found numerous applications in deep learning systems. In this context, graph based methods have been used to generalize classical CNNs to irregular data domains, with GNNs achieving state-of-the-art results in a number of applications [37]. On the other hand, tensor methods have been used to relax the computational complexity of neural networks [28], as well as to alleviate their notorious "black-box" nature [30, 74]. These promising results have also highlighted a void in literature regarding the combination of both techniques in order to solve deep learning challenges, especially in the area of sequence modelling. To this end, we introduce a novel RGTN framework for multi-way time-series modelling, which enhances the sequence modelling ability of RNNs [69] through tensor- and graph-theoretic concepts.

The field of GSP generalizes traditional signal processing concepts to irregular domains [35, 34, 36], which are naturally represented as graphs. Developments in GSP have led to a range of spatial and spectral based techniques that generalize the notion of frequency and locality to irregular data, allowing for the processing of signals while taking into account the underlying data domain [33]. Several concepts

developed in GSP have found applications in deep learning, where graph filters can be implemented across multiple GNN layers to incorporate graph topology information [37].

Tensors are multi-linear generalization of vectors and matrices to multi-way arrays, which allows for a richer representation by not limiting the data to the classical "flat-view" matrix approaches [17]. Recent developments in tensor manipulation have led to TD techniques that can represent high dimensional tensors through a contracting network of smaller core tensors. Such TD techniques can be used to compress the number of parameters needed to represent high-dimensional data, and have already found applications in deep learning. Notably, it has been shown that TD techniques, such as the TTD [49], can be used to compress neural networks considerably while maintaining comparable performance [28, 29, 76].

However, despite promising results achieved in both individual fields, the full potential arising from the combination of graphs, tensors, and neural networks is yet to be explored, especially in the area of sequence modelling. To this end, we set out to investigate the extent to which a careful domain consideration of tensors and graphs can improve the complexity and performance of RNNs, by leveraging the theoretical frameworks underpinning graph ML and TNs. More specifically, we establish a novel structure for the modelling of RNN hidden states through a multi-linear graph filter embedded in a TN architecture, leading to a novel RGTN framework. The so derived RGTN exploits both the ability of graphs to process data defined on irregular time-domains and the expressive power of TD, resulting in a new class of expressive models with drastically lower complexity compared to standard RNNs. Our experimental results confirm the superiority of the proposed RGTN models, demonstrating desirable properties in terms of both performance and complexity across several multi-way sequence modelling tasks.

9.2 Preliminaries

9.2.1 *Recurrent Neural Networks*

RNNs [69] [144] are among the most successful deep learning tools for sequence modelling. A standard RNN layer captures time-varying dependencies by processing hidden states, $\boldsymbol{h}_t \in \mathbb{R}^M$, at time t through feedback (or recurrent) weights as:

$$\boldsymbol{h}_t = \sigma_h(\boldsymbol{W}^{(h)}\boldsymbol{h}_{t-1} + \boldsymbol{W}^{(x)}\boldsymbol{x}_t + \boldsymbol{b}^{(h)}) \tag{9.1}$$

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

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

$$\boldsymbol{y}_t = \sigma_y(\boldsymbol{W}^{(y)}\boldsymbol{h}_t + \boldsymbol{b}^{(y)}) \tag{9.2}$$

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

9.3 Recurrent Graph Tensor Networks

9.3.1 *General Recurrent Graph Tensor Networks*

Consider the RNN forward pass in (9.1) without the optional bias vector and activation function:

$$\boldsymbol{h}_t = \boldsymbol{W}^{(h)}\boldsymbol{h}_{t-1} + \boldsymbol{W}^{(x)}\boldsymbol{x}_t \tag{9.3}$$

Denote $\hat{\boldsymbol{x}}_t = \boldsymbol{W}^{(x)}\boldsymbol{x}_t \in \mathbb{R}^M$, for $t = 1, \ldots, \tau$ time-steps; then (9.3) can be written in a block-matrix form:

$$\begin{bmatrix} \boldsymbol{h}_\tau \\ \boldsymbol{h}_{\tau-1} \\ \vdots \\ \boldsymbol{h}_1 \end{bmatrix} = \begin{bmatrix} (\boldsymbol{W}^{(h)})^0 & (\boldsymbol{W}^{(h)})^1 & \cdots & (\boldsymbol{W}^{(h)})^{\tau-1} \\ 0 & (\boldsymbol{W}^{(h)})^0 & \cdots & (\boldsymbol{W}^{(h)})^{\tau-2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & (\boldsymbol{W}^{(h)})^0 \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}}_\tau \\ \hat{\boldsymbol{x}}_{\tau-1} \\ \vdots \\ \hat{\boldsymbol{x}}_1 \end{bmatrix} \tag{9.4}$$

We now define: (i) $\boldsymbol{X} \in \mathbb{R}^{\tau \times N}$, as the input matrix generated by stacking row-vectors, $\boldsymbol{x}_t$, over τ successive time-steps; (ii) $\hat{\boldsymbol{X}} \in \mathbb{R}^{\tau \times M}$, as $\hat{\boldsymbol{X}} = \boldsymbol{X} \times_2^2 \boldsymbol{W}^{(x)}$; (iii) $\boldsymbol{H} \in \mathbb{R}^{\tau \times M}$, as the matrix generated by stacking hidden state vectors, $\boldsymbol{h}_t$, as row-vectors over τ time-steps; and (iv) $\boldsymbol{R} \in \mathbb{R}^{\tau M \times \tau M}$, as the block matrix composed by the powers of $\boldsymbol{W}^{(h)}$ from (9.4). This allows (9.4) to be expressed compactly as:

$$\mathrm{vec}(\boldsymbol{H}) = \boldsymbol{R} \times_2^1 \mathrm{vec}(\hat{\boldsymbol{X}}) \tag{9.5}$$

Without loss of generality, we shall further restrict the feedback matrix, $\boldsymbol{W}^{(h)}$, to be a scaled idempotent matrix, that is $\boldsymbol{W}^{(h)} = c\boldsymbol{W}^{(r)}$, where c is a positive scaling constant strictly less than 1, and $\boldsymbol{W}^{(r)}$ is an idempotent matrix that models how information propagates between successive time-steps. For this setup, the feedback matrix has the property $(\boldsymbol{W}^{(h)})^p = c^p\boldsymbol{W}^{(r)}$, for $p > 0$. This allows the block matrix $\boldsymbol{R}$ to be decomposed as:

$$\boldsymbol{R} = \boldsymbol{I} + \boldsymbol{A} \otimes \boldsymbol{W}^{(r)} \tag{9.6}$$

where $\boldsymbol{A} \in \mathbb{R}^{\tau \times \tau}$ contains the constants c^p, as:

$$\boldsymbol{A} = \begin{bmatrix} 0 & c^1 & \cdots & c^{\tau-1} \\ 0 & 0 & \cdots & c^{\tau-2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{9.7}$$

Note that the matrix, $\boldsymbol{A}$, can be interpreted as the weighted graph adjacency matrix connecting τ successive time-steps as vertices (nodes). This also justifies its triangular (directed) nature, since only past information can influence future states but not vice-versa.

We now denote, $\boldsymbol{\mathcal{R}} \in \mathbb{R}^{\tau \times M \times \tau \times M}$, as the 4-th order tensorization of $\boldsymbol{R}$, that is $\boldsymbol{\mathcal{R}} = \mathrm{ten}(\boldsymbol{I} + \boldsymbol{A} \otimes \boldsymbol{W}^{(r)})$; this allows us to simplify the expression in (9.5) via a double tensor contraction, and express the general RGTN filtering operation in its complete form as:

$$\boldsymbol{H} = \boldsymbol{\mathcal{R}} \times_{3,4}^{1,2} \boldsymbol{X} \times_2^2 \boldsymbol{W}^{(x)} \tag{9.8}$$

The proposed filtering operation in (9.8) can be used to extract features from time-series data, $\boldsymbol{X}$, in a neural network. We will refer to such neural network models as general RGTNs (gRGTN).

9.3.2 *Simplified Recurrent Graph Tensor Networks*

To establish a link between the proposed RGTN filtering operation and classical spatial graph filters, we shall now consider a special case of equation (9.8).

Consider a special case where $\boldsymbol{W}^{(r)} = \boldsymbol{I}$. This implies that $\boldsymbol{W}^{(h)} = c\boldsymbol{I}$, which simplifies the hidden state evolution in (9.3) as $\boldsymbol{h}_t = c\boldsymbol{h}_{t-1} + \hat{\boldsymbol{x}}_t$. This corresponds a simplified system model where the past information is propagated to the future with a scaling constant of c. This simplifies (9.8) as:

$$\begin{aligned}\boldsymbol{H} &= \boldsymbol{\mathcal{R}} \times_{3,4}^{1,2} \boldsymbol{X} \times_2^2 \boldsymbol{W}^{(x)} \\ &= \text{ten}(\boldsymbol{I} + \boldsymbol{A} \otimes \boldsymbol{W}^{(r)}) \times_{3,4}^{1,2} (\boldsymbol{X} \times_2^2 \boldsymbol{W}^{(x)}) \\ &= \text{ten}(\boldsymbol{I} + \boldsymbol{A} \otimes \boldsymbol{I}) \times_{3,4}^{1,2} (\boldsymbol{X} \times_2^2 \boldsymbol{W}^{(x)}) \\ &= (\boldsymbol{I} + \boldsymbol{A}) \times_2^1 (\boldsymbol{X} \times_2^2 \boldsymbol{W}^{(x)}) \\ &= (\boldsymbol{I} + \boldsymbol{A}) \times_2^1 \hat{\boldsymbol{X}}\end{aligned} \tag{9.9}$$

Notice that (9.9) is equivalent to $\boldsymbol{H} = \sum_{k=0}^{K-1} \alpha_k \boldsymbol{A}^k \hat{\boldsymbol{X}}$, which is precisely a spatial graph filter as discussed in Section 2.2, where $K = 2$, $\alpha_k = 1$, and $\boldsymbol{A}$ is the weighted graph adjacency matrix that enforces the directed flow of time. We will refer to neural networks employing equation (9.9) for feature extraction as simplified RGTNs (sRGTN).

9.3.3 *Tensor Network Formulation*

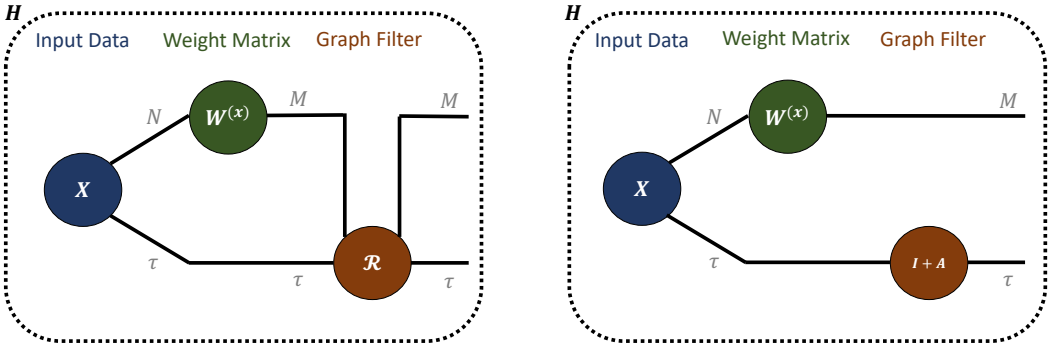


Figure 9.1: TN diagram of the gRGTN filtering operation (left) according to (9.8), and the sRGTN filtering operation (right) according to (9.9). The nodes of the TN diagram represent different tensors, while the edges represent tensor contractions over common dimensions between tensors. The dimensions of different tensors are denoted in gray letters.

Consider the gRGTN filtering operation in (9.8). The multi-linear nature of the tensor $\boldsymbol{\mathcal{R}}$ and the associated double tensor contraction naturally admits a TN representation, as shown in Figure 9.1 (left). Similarly, the sRGTN filtering in (9.9) also admits a TN representation with a simpler topology, as shown in Figure 9.1 (right). This allows the hidden state modelling operation to benefit from the enhanced

expressive power of tensors, which are not limited to the standard "flat-view" matrix methods [18, 19].

By integrating the concept of graph filtering in a TN framework, we can easily design network architectures for processing time-series data of any modalities, as well as leverage on the power of TD to boost its expressive power while maintaining low complexity. For illustration, Figure 9.3 shows TN models designed to process multi-way time series data as order-3 input tensors (i.e. the time-series features are indexed along a time-mode and an additional physical mode), which uses appropriate TT networks to process filtered multi-way time-series data.

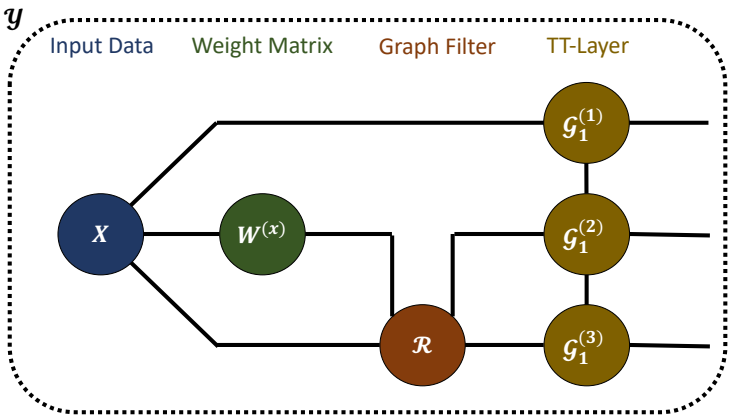


Figure 9.2: A gRGTN model designed to handle order-3 input data tensors. In addition to the proposed RGTN filtering operation, it leverages the power of TTD networks (in yellow) to achieve high expressive power at low parameter complexity, which is inherently compatible with the multi-way nature of the gRGTN.

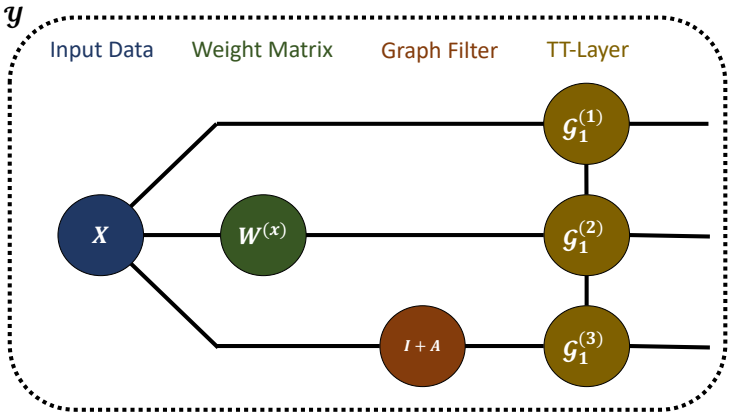


Figure 9.3: An sRGTN model designed to handle order-3 input time-series tensors. Similar to gRGTN, it leverages on the power of TTD networks (in yellow) to achieve high expressive power at low parameter complexity, which is inherently compatible with the multi-way nature of the RGTN framework.

Remark 23 *The double tensor contraction with $\mathcal{R}$ in gRGTN implies a stronger coupling of features with the underlying time-domain represented in graph form, thus yielding enhanced expressive power over the decoupled contractions in sRGTN.*

Remark 24 *The TT layers in Figure 9.3 can be interpreted as tensorized fully-connected neural network layers compressed via TTD, which drastically reduces the number of parameters required to achieve the same expressive power [28, 29].*

9.4 Experiments

9.4.1 *Datasets*

To validate the expressive power of the proposed gRGTN and sRGTN models, we verified their performance in a number of multi-way time-series modelling tasks, including:

1. Beijing Multi-Site Air Quality Forecasting [130]. This dataset consists of various air quality measurements obtained across 12 different sites in China recorded at an hourly rate. The learning task for this dataset is to forecast the air quality level across all 12 sites in the next hour.

2. Global Land Temperature Forecasting [145]. This dataset consists of monthly temperature recordings obtained across multiple cities around the world. The learning task for this dataset is to forecast the average temperature across 14 major cities in India during the next month.

3. Liverpool House Price Forecasting [146]. This dataset consists of monthly price indices across 4 different types of houses in Liverpool, United Kingdom. The learning task for this dataset is to forecast the price indices of different house types in the next month.

4. Multi-Sensor Activity Recognition [147]. This dataset consists of multi-sensor measurements of human bodies when performing different physical activities. The learning task for this dataset is to classify the physical activity from the multi-sensor measurements.

All of the considered data are multi-modal time-series tensors of order-3. The exact modalities of the input data tensors are summarized in Table 9.1.

Table 9.1: Experiment Data Modalities

	Mode 1		Mode 2		Mode 3	
	Physical Mode	Size	Physical Mode	Size	Physical Mode	Size
Air Quality	Time	6	Site	12	Features	27
Temperature	Time	6	City	14	Features	4
House Price	Time	6	House Type	4	Features	2
Activities	Time	24	Sensor	3	Features	3

9.4.2 *Benchmark Models and Metrics*

We compared the performance of the proposed gRGTN and sRGTN models against standard RNN, GRU, and LSTM based neural networks. For comparable results, all models have the exact same model architecture, hidden units, activation functions, and training method, with the only differences being: (i) the feature extraction layer used, which can be based on gRGTN, sRGTN, RNN, GRU, or LSTM, and (ii) the fully-connected dense layers, which are replaced by the equivalent TT networks for gRGTN and sRGTN as shown in Figure 9.3 [28]. For more details, please refer to the full experiment code provided on GitHub[1].

We compared the considered models across the proposed experiments both in terms of performance and complexity. In terms of performance metrics, we used out-of-sample Mean Absolute Error (MAE) for the regressions tasks related to datasets (1), (2), and (3), and classification accuracy for the classification task related to dataset (4). In terms of complexity, we compare the number of trainable parameters needed to achieve the same model specifications.

9.4.3 *Experiment Results*

The experiment results are summarized in Table 9.2. The top table shows the test set performance for three regression tasks (measured in MAE) and one classification task (measured in accuracy) achieved by the considered models. The bottom table shows the corresponding number of trainable parameters needed for each task.

By virtue of its graph and tensor structure, the proposed gRGTN model achieved the best performance overall, obtaining the highest score for 2 out of 4 datasets, while using drastically less number of trainable parameters compared to standard RNN, GRU, and LSTM models. On the other hand, the sRGTN model achieved the lowest parameter complexity due to its approximation assumption of $\boldsymbol{W}^{(r)} = \boldsymbol{I}$, but at the cost of marginally reduced performance.

Table 9.2: Performance of the Considered Models.

Test Set Score	gRGTN	sRGTN	RNN	GRU	LSTM
Air Quality Forecasting (MAE)	**0.01598**	0.01742	0.01872	0.01706	0.01652
Temperature Forecasting (MAE)	0.20959	0.27491	0.19905	0.18050	**0.17744**
House Price Forecasting (MAE)	0.72946	0.76768	**0.71195**	0.74081	0.73463
Activity Classification (Accuracy)	**79.883%**	78.740%	50.731%	79.398%	78.629%

Table 9.3: Complexity of the Considered Models

Number of Trainable Parameters	gRGTN	sRGTN	RNN	GRU	LSTM
Air Quality Forecasting	556	**492**	2844	8196	10836
Temperature Forecasting	406	**342**	718	1782	2278
House Price Forecasting	220	**156**	244	540	652
Activity Classification	301	**237**	261	573	693

[1]The code is available on www.github.com/gylx/RGTN

9.5 Further Experiments on Financial Data

9.5.1 *Preliminaries on Limit Order Books*

Limit order books (LOBs) play a central role in the formation of prices for financial securities in exchanges globally. These systems gather limit orders to buy or sell certain prices and sizes of securities from large numbers of dealers and investors, centralizing the orders in a public, electronic registry to match bids and offers in an automatic, transparent process. This function makes the understanding of order book dynamics especially interesting to model. The aggregation of prices and sizes for buy and sell orders across various levels results in non-stationary time series of high dimensions and multimodal nature, which poses a challenge for standard ML models. In addition, scaling up to higher dimensions implies an exponential increase in the number of parameters required, besides the associated increase of computational complexity.

LOB level	Ask price	Ask size	Bid price	Bid size
1	585.94	200	585.33	18
...	...	...	...	...
10	587.90	500	584.27	300

Table 9.4: Sample LOB state structure with 40 dimensions for a given timestamp of AAPL stock on June 26, 2012. Source: `lobsterdata.com`

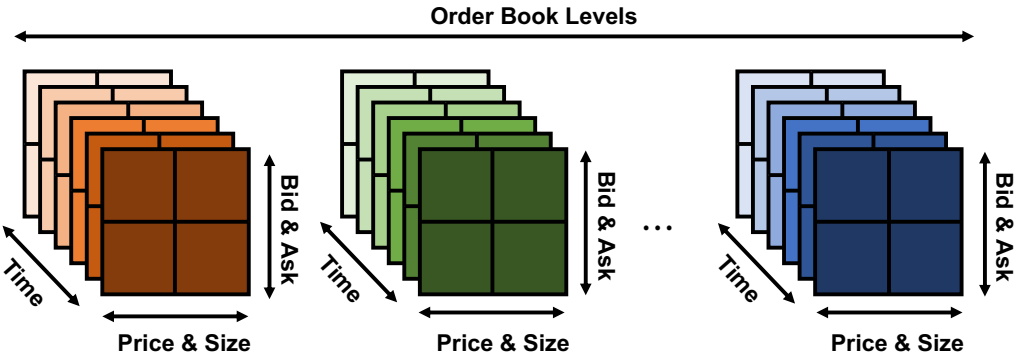


Figure 9.4: Tensor structure of the order book. The order book data of a stock at a given point in time contains the buy side (bid) and sell side (ask) information about pricing and the size. The l-th best bid and ask available are contained within the l-th level of the order book. By considering the order book information across L levels over T time steps, we obtain order-4 data tensors of [time] $\times$ [price and size] $\times$ [bid and ask] $\times$ [order book levels].

9.5.2 *Experiment Overview*

The purpose of our experiments is to investigate the added value of including graph components in the tensorized RNN (producing an RGTN) when modelling the time series of LOB data. Series of LOB states tend to be of high dimension, given the various levels of depth from the best bid (order to buy) and best ask (order to sell) in the book, with every level containing price and size for bid and ask. We consider a LOB of $L = 10$ levels, whose input dimension is therefore 40 (bid/ask prices and bid/ask sizes for each level). Finally, as customary in modelling sequential data with RNNs, to build the training sample for each timestep with the N last data points available we use a look-back window of $N = 20$, resulting in an input dimension of $D = 800$, where the added value of the TT structure becomes self-evident.

In order to evaluate the performance of each model on each stock over 40 dimensions, we considered a specific statistic to summarize the state of the order book at each timestamp: the Volume Weighted Average Price (VWAP) derived from the order book of each stock, as defined in equation (9.10). This quantity defines the average price for each stock weighted against the order sizes available for each price across the order book, thus summarizing the state of the supply and demand liquidities. Considering L levels of the order book, the VWAP at time t is defined as:

$$VWAP_L(t) = \frac{\sum_{l=1}^{L} p_l(t)V_l(t)}{\sum_{l=1}^{L} V_l(t)} \tag{9.10}$$

where $p_l(t)$ and $V_l(t)$ are the price and the size available at l-th depth at time t. In particular, we tested the ability of each model (RGTN, TTRNN and GRU) to learn the VWAP of each order book across 5 different stocks independently.

9.5.3 *Dataset*

We trained and tested our models on LOB sample data supplied by LOBSTER, an online LOB data tool which provides easy-to-use, high-quality LOB data for academic research purposes (`lobsterdata.com`). The data consist of two types of input files:

- **Messages** containing the arrival of specific events that update the state of the order book, such as new orders, modifications, cancellations, etc. and their respective price, size and side (buy or sell). Timestamps are defined per seconds after midnight, down to microsecond precision. Sizes are number of shares, and prices represent USD prices times 10,000. Sides take either buy (+1) or sell (-1) values.

- **Order books** contain the evolution of the state of the order book for up to $L = 10$ levels of prices and sizes, updated per the events that arrive through the order messages.

Timestamp	OrderID	Size	Price	Side	Stock
34200.0042	16113575	18	5853300	+1	AAPL
34200.0255	16120480	10	5859200	-1	AAPL
...	...	...	...	...	...
57599.9131	287150868	48	5776100	-1	AAPL

Table 9.5: Sample of LOB messages for AAPL

In particular, it contains LOB data series for five stocks (Apple, Amazon, Google, Intel, Microsoft) during the trading session of June 26, 2012 which consists of the registry of order messages for the day, and the sequence of LOB states that derives from such order messages up to 10 levels of bids and offers, respectively:

Stock	Ticker	Messages	Time span
Apple	AAPL	400,391	9:30:00.00 - 15:59:59.91
Amazon	AMZN	269,748	9:30:00.02 - 15:59:59.96
Google	GOOG	147,916	9:30:00.02 - 15:59:59.87
Intel	INTC	624,040	9:30:00.01 - 15:59:59.95
Microsoft	MSFT	668,765	9:30:00.01 - 15:59:59.91

Table 9.6: Order inputs for each stock considered.

The LOB states consist of asynchronous time series, updated upon arrival of new orders with random distributions in time. For each stock series, we compute the value of VWAP following Eq. (9.10) for every second, sampling the state of the order book on that timestamp. Data from messages containing price and size orders is implicit in the LOB states series as the former imply the changes in the latter.

9.5.4 *Benchmark Models and Metrics*

The experiments compared three models:

- **GRU** the mainstream GRU class of RNN, as implemented in Keras TensorFlow 2, serves as benchmark of performance with respect to the two TNs.
- **TTRNN** a RNN structured to process a tensor input through kernels defined by the sequential contraction of core tensors, as discussed in Chapter 4. This allows for a substantial reduction in the number of parameters required.
- **RGTN** a TN including a time-domain graph adjacency matrix for the extraction of time-series features in the forward pass, as discussed in Section 9.3.2.

Each of the three neural models was trained independently over 100 epochs for each of the five stocks listed in Table 9.6, resulting in 15 experiments. The training data split consists of 70% of the LOB data (of which 20% was used for validation), and the remaining 30% to test the outputs of the trained models. Each experiment was then repeated 30 times with different random seeds. Weights were initialized

with the default Xavier method in `tf.keras`. Table 9.7 summarizes the dimensional structure of each model.

Layer	Input Shape	Output Shape	Parameters
GRU	20×40	40	9840
Dense	40	1	41

GRU Model summary.

Layer	Input Shape	Output Shape	Parameters
TTRNN	$20 \times 1 \times 10 \times 2 \times 2$	$1 \times 1 \times 10 \times 2 \times 2$	852
Flatten	$1 \times 1 \times 10 \times 2 \times 2$	40	0
Dense	40	1	41

TTRNN Model summary.

Layer	Input Shape	Output Shape	Parameters
RGTN	$20 \times 1 \times 10 \times 2 \times 2$	$1 \times 1 \times 10 \times 2 \times 2$	109
Flatten	$1 \times 1 \times 10 \times 2 \times 2$	40	0
Dense	40	1	41

RGTN Model summary.

Table 9.7: Model summaries for 40-dimensional LOB states times 20-timestep lookback, with number of trainable parameters and scalar output dimension for VWAP.

The performance was evaluated through various metrics, such as the mean standard error (MSE) of the output, its standard deviation, the average training time and the number of trainable parameters required for each model and stock considered. Results demonstrate a consistent superior performance of our proposed model across the considered five cases for each of these metrics. Next sections develop the details of the data used, each of the models, the success metrics and their conclusions.

9.5.5 *Experiment Results*

The outcomes of the experiments were consistent across the five stocks studied with respect to the advantages of employing the RGTN model structure. Table 9.8 summarizes the results for all the metrics considered in each case. The experimental results obtained were as follows:

- **Mean Standard Error**. Both tensor-based models outperformed the GRU with three- to four-fold improvements in the MSE when modelling the VWAP; the MSE for the RGTN was also slightly better than TTRNN across the five stocks. The MSE for each experiment was the average MSE obtained across the 30 independent repetitions of the experiment with different random seeds.

- **Variance**. The variance of the test-MSE across the 30 repetitions of each experiment is the variable that shows the greatest improvement when using the RGTN instead of the TTRNN, with improvements of around one order of

magnitude of standard deviations with respect to the TTRNN model on average, and far superior to those of the GRU across all the stocks considered. Figure 9.5 displays the MSE distributions compared across all tensor models.

- **Training time**. The TTRNN took the longest to train, with well above 4h in all cases. While the GRU nearly halves that time, the RGTN concluded its training within 40 minutes in all cases, implying between five- and nine-fold training time improvements.
- **Trainable parameters**. Reducing the number of trainable parameters is the core motivation for the use of TNs in high-dimensional problems, and the TTRNN structures required in our experiments barely 9% the number of trainable parameters of the GRU counterpart. Including the graph component results in a further improvement, where the RGTN required around 17% of the trainable parameters of the TTRNN (1.52% of those of the GRU).
- **Gradients**. As discussed in previous sections, a shortcoming of TTRNNs is the phenomenon of exploding or vanishing gradients. We observed various such cases while training the TTRNNs for these experiments, while none of the RGTN cases exhibited such problematic gradients. This reinforces further the case for the use of RGTNs instead of TTRNNs for data and experiments of this nature. See Figure 9.5 for details of the gradient evolution for each experiment.

9.6 Summary

We have introduced a novel RGTN framework for modelling time-series data, by combining the expressive power of TNs with the ability of graphs to account for the structure underlying time-series data. Experiment results have verified the desirable properties of the proposed RGTN framework, which outperformed RNN, GRU, LSTM, and TTRNN models across multiple time-series modelling tasks, and at a drastically reduced parameter complexity.

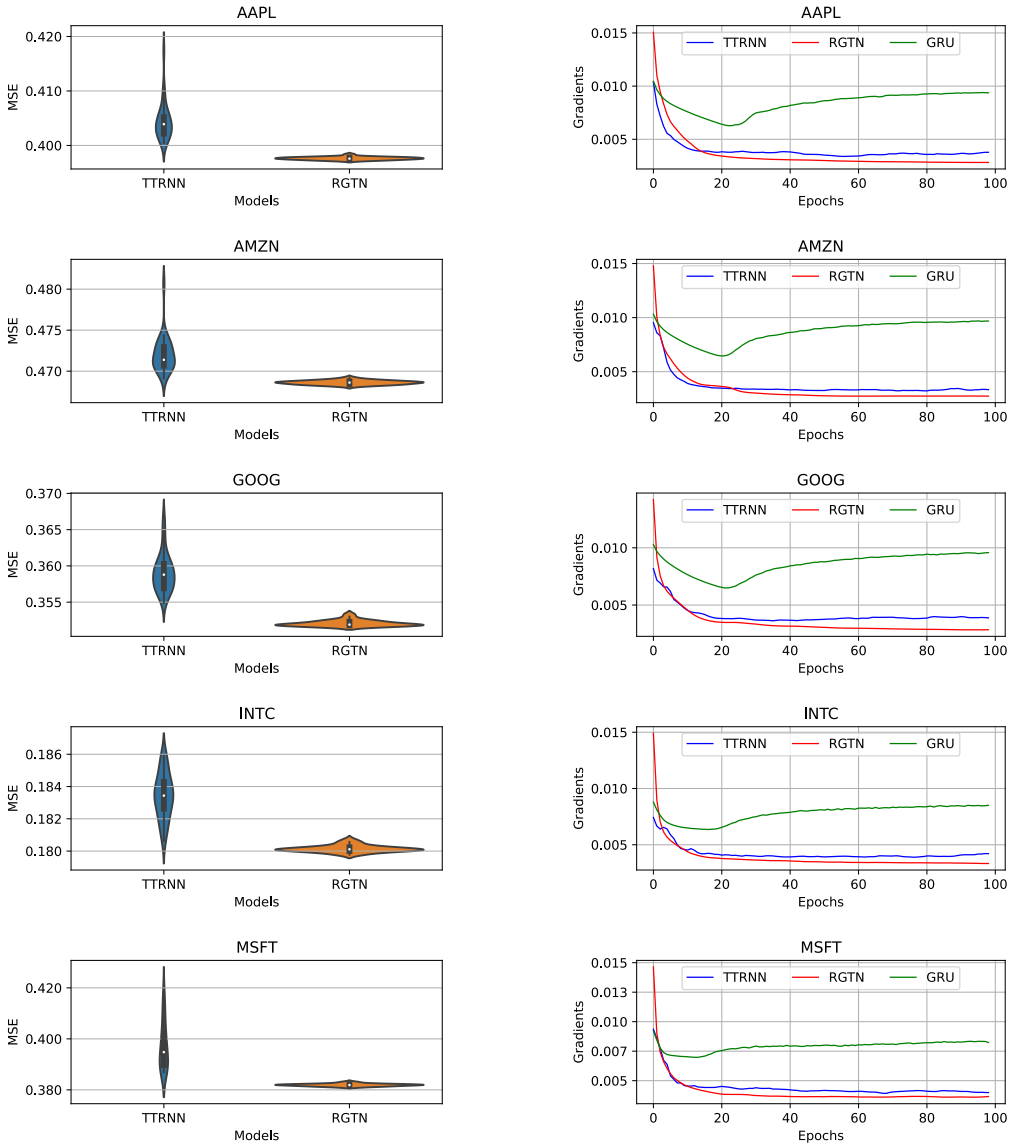


Figure 9.5: Experiment results. (Left) MSEs and their variances: The RGTN models provide lower MSEs and smaller output variances than TTRNN for each of the five stocks considered. Both outperform significantly the GRU benchmark model. (Right) Gradients: Evolution of gradients of each model considered across training epochs. Tensorized models display better behavior, with fastest convergence of gradients in each instance of RGTN experiments.

Model	MSE	Training time	Parameters
GRU	1.4950 ± 0.0648	2:33:13	9881
TTRNN	0.4045 ± 0.0033	4:30:49	893
RGTN	0.3976 ± 0.0003	0:36:03	150

Results for AAPL

Model	MSE	Training time	Parameters
GRU	1.7814 ± 0.0845	2:32:12	9881
TTRNN	0.4720 ± 0.0021	4:33:03	893
RGTN	0.4686 ± 0.0002	0:37:41	150

Results for AMZN

Model	MSE	Training time	Parameters
GRU	1.5374 ± 0.0850	2:32:47	9881
TTRNN	0.3590 ± 0.0026	4:31:17	893
RGTN	0.3521 ± 0.0004	0:35:46	150

Results for GOOG

Model	MSE	Training time	Parameters
GRU	0.5703 ± 0.1297	2:33:40	9881
TTRNN	0.1834 ± 0.0014	4:30:38	893
RGTN	0.1801 ± 0.0002	0:36:41	150

Results for INTC

Model	MSE	Training time	Parameters
GRU	1.2437 ± 0.1899	2:34:08	9881
TTRNN	0.3972 ± 0.0093	4:31:09	893
RGTN	0.3819 ± 0.0005	0:36:26	150

Results for MSFT

Table 9.8: Summary of experimental results across models and stocks: MSE within a standard deviation, training time, and number of trainable parameters.

Chapter 10

Attention Graph Tensor Networks

Building upon the RGTNs introduced in the previous chapter, this chapter explores their application in the context of attention modelling. The attention mechanism is at the core of state-of-the-art NLP models, owing to its ability to focus on the most contextually relevant part of a sequence. However, current attention models rely on "flat-view" matrix methods to process sequence of tokens embedded in vector spaces, resulting in exceedingly high parameter complexity for practical applications. To this end, we introduce a novel TSA mechanism, which leverages on the GTN framework to efficiently process tensorized token embeddings via attention based spectral graph filters. By virtue of multi-linear algebra, such tensorized token embeddings are shown to effectively bypass the curse of dimensionality, reducing the parameter complexity of the attention mechanism from exponential to linear in the weight matrix dimensions. Furthermore, the graph formulation of the attention domain enables the processing of tensorized embeddings through spectral graph convolution filters, which further increases its expressive power. The benefits of the TSA are demonstrated through five benchmark NLP experiments, where the proposed mechanism is shown to achieve better or comparable results against traditional attention models, while incurring drastically lower parameter complexity.

10.1 Introduction

The attention mechanism has become the *de-facto* tool of choice for NLP tasks, owing to its ability to capture context dependent relationships in data and thus focus on the most relevant part of a sequence [148]. Despite being at the core of many state-of-the-art language models [149, 150, 151], the attention mechanism is known to suffer from the curse of dimensionality, whereby the parameter complexity increases exponentially with the size of the associated weight matrices [152]. To alleviate the complexity issues in attention models, we introduce the TSA mechanism. The proposed TSA builds on a recent GTN framework [16] [11], which drastically reduces the complexity costs through TD techniques, while at the same time boosting the expressive power of the attention operation via spectral graph convolutions.

Contributions: The contributions of this work are threefold. First, we generalize the attention mechanism to make it possible to process token embeddings in a higher-order tensor space, as opposed to the traditional vector space embeddings. By virtue of its multi-modal structure, such a tensorized embedding benefits from the power of multi-linear algebra and the associated TD algorithms, hence bypassing the curse of dimensionality aspect associated with traditional NLP methods. Second, we introduce a novel formulation of the attention mechanism as a graph filter operation, which enables the processing of tensorized embeddings through spectral graph convolution filters. Third, a unified GTN framework is presented that integrates the advantages of both tensors and graphs in the context of attention, resulting in a highly expressive attention model with drastically low complexity costs.

Related Work: TTD [49] has been used to compress embedding layers for NLP tasks with large vocabularies in [153], although it has not been employed to compress the attention mechanism directly. The Khatri-Rao product was employed to develop a tensor based attention with lower complexity in [154], but it focuses on dimension-wise attention instead of token-wise attention, and is limited to only a low-dimensional order-3 tensor for processing vector embeddings. Block-Term TD [155] has also been used to propose a multi-linear attention mechanism in [152], but it was still limited to a low-dimensional order-3 tensor for processing vector embeddings, and could only approximate the classical attention mechanism if the embedding dimension is equal to the length of the sequence. The attention mechanism has been applied to GNNs both in the spatial and spectral domains [134, 156, 157], but the attention operation was used in a masking context with a priori defined graph edges, and the application was limited to semi-supervised learning. Overall, to the best of our knowledge, our work is the first to extend the attention mechanism to process tensor embeddings of arbitrarily high dimensions, by leveraging on the expressive power of TNs and spectral graph convolutions.

Chapter Organization: In Section 10.2, we introduce the theoretical background necessary for this work. A graph formulation of the spectral attention mechanism is next introduced in Section 10.3, which is then generalised to the tensor domain in Section 10.4. All of the derived concepts are then combined in Section 10.5. We finally validate the proposed framework via extensive experiments in Section 10.6 and conclude our findings in Section 10.7.

10.2 Preliminaries

10.2.1 *Dot Product Attention*

Query, Key, Value: Given an input time-series matrix, $\boldsymbol{X} \in \mathbb{R}^{L \times I}$, where I features are indexed along L time-steps, the Dot Product Attention (DPA) [148] computes: (i) $\boldsymbol{Q} \in \mathbb{R}^{L \times J}$, a query-representation of the input data generated by the transform, $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^{(q)}$, where $\boldsymbol{W}^{(q)} \in \mathbb{R}^{I \times J}$ is a trainable query-weight matrix; (ii) $\boldsymbol{K} \in \mathbb{R}^{L \times J}$, a key-representation of the input data generated by the transform, $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{(k)}$, where $\boldsymbol{W}^{(k)} \in \mathbb{R}^{I \times J}$ is a trainable key-weight matrix; (iii) $\boldsymbol{V} \in \mathbb{R}^{L \times J}$, a value-representation of the input data generated by the linear transform, $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^{(v)}$, where $\boldsymbol{W}^{(v)} \in \mathbb{R}^{I \times J}$ is a trainable value-weight matrix.

Attention Coefficients: Given $\boldsymbol{Q}$ and $\boldsymbol{K}$, the self-attention coefficient matrix, $\boldsymbol{\Theta} \in \mathbb{R}^{L \times L}$, is computed as $\boldsymbol{\Theta} = \sigma(\frac{1}{\sqrt{d_k}}\boldsymbol{Q}\boldsymbol{K}^T)$, where $\sigma(\cdot)$ is a $softmax$ activation function and $\frac{1}{\sqrt{d_k}}$ is a scaling factor. More specifically, the attention coefficient between the l_1-th and l_2-th time-step, $\boldsymbol{\Theta}_{l_1,l_2}$, is defined as the inner product between the l_1-th query vector, $\boldsymbol{q}_{l_1}$, and l_2-th key vector, $\boldsymbol{k}_{l_2}$, as $\boldsymbol{\Theta}_{l_1,l_2} = \sigma(\frac{1}{\sqrt{d_k}}\boldsymbol{q}_{l_1}^T\boldsymbol{k}_{l_2})$.

Multi-Head Attention: Given $\boldsymbol{\Theta}$, the attention mechanism then generates the final output, $\boldsymbol{Y} \in \mathbb{R}^{L \times J}$, as $\boldsymbol{Y} = \boldsymbol{\Theta}\boldsymbol{V}$. For multi-head attention, this can be repeated H times using a set of attention coefficient matrices, $\{\boldsymbol{\Theta}_1, \ldots, \boldsymbol{\Theta}_H\}$, to generate H different outputs, $\{\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_H\}$, that are then concatenated and combined to produce the final output, $\boldsymbol{Y}$.

10.2.2 *Graph Tensor Networks*

Recurrent Graph Tensor Network: Consider a sequence learning problem where the input data, $\boldsymbol{X} \in \mathbb{R}^{L \times J}$, is a time-series matrix with J features indexed along L time-steps. For a so defined input matrix, an RGTN [11] extracts time-series features, $\boldsymbol{Y} \in \mathbb{R}^{L \times J}$, by applying a tensor contraction based forward pass, $\boldsymbol{Y} = \text{ten}(\boldsymbol{I} + \boldsymbol{A} \otimes \boldsymbol{P}) \times_{3,4}^{1,2} \boldsymbol{X}$, where $\boldsymbol{I} \in \mathbb{R}^{LJ \times LJ}$ is an identity matrix, $\boldsymbol{P} \in \mathbb{R}^{J \times J}$ is an idempotent matrix modelling the propagation of information between successive time-steps, and $\boldsymbol{A} \in \mathbb{R}^{L \times L}$ is a time-domain graph adjacency matrix. More specifically, the time-adjacency matrix, $\boldsymbol{A}$, considers each of the L time-steps as a node of a time-domain graph, where the l_1-th and l_2-th time-steps are connected with weight, $\boldsymbol{A}_{l_1,l_2}$, such that $\boldsymbol{A}_{l_1,l_2} = c^{l_2 - l_1}$ for $l_2 > l_1$, and $\boldsymbol{A}_{l_1,l_2} = 0$ otherwise, where c is a damping constant strictly less than 1. The condition $l_2 > l_1$ ensures that $\boldsymbol{A}$ is an upper-triangular matrix to reflect the structured flow of information over time, as past information can influence the future states but not vice-versa.

Simplified Recurrent Graph Tensor Network: A low complexity version of RGTN was also proposed in [11] by approximating $\boldsymbol{P} \approx \boldsymbol{I}$, which simplifies the RGTN forward pass to $\boldsymbol{Y} = (\boldsymbol{I} + \boldsymbol{A}) \times_2^1 \boldsymbol{X}$. For multi-modal input features, the RGTN

forward pass can be easily extended to tensor form, $\boldsymbol{\mathcal{Y}} = (\boldsymbol{I} + \boldsymbol{A}) \times_2^1 \boldsymbol{\mathcal{X}}$, where $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{L \times J_1 \times \cdots \times J_N}$ is a time-series tensor of order-$(N+1)$, which contains features of dimensionality $J_1 \times \cdots \times J_N$ for each of the L time-steps.

Bi-Directional Graph Tensor Network: The time-adjacency matrix, $\boldsymbol{A}$, was originally developed to model sequential data. However, more generally, it can be replaced by other adjacency matrices depending on the demands of an application [16]. For instance, for time-series applications where the directed flow of information over time is not necessary, we can replace $\boldsymbol{A}$ with a bi-directional time-graph adjacency matrix, $\boldsymbol{\Omega} \in \mathbb{R}^{L \times L}$, defined as $\boldsymbol{\Omega} = \frac{1}{2}(\boldsymbol{A}^T + \boldsymbol{A})$.

10.3 Spectral Attention

10.3.1 *Time-Domain Attention Graph*

We shall now introduce the notion of the time-domain attention graph. Given an input time-series matrix, $\boldsymbol{X} \in \mathbb{R}^{L \times I}$, where I features are indexed along L time-steps, we compute: (i) $\boldsymbol{K} \in \mathbb{R}^{L \times J}$, a key-representation of the input data generated by the transform, $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{(k)}$, where $\boldsymbol{W}^{(k)} \in \mathbb{R}^{I \times J}$ is a trainable key-weight matrix; (ii) $\boldsymbol{V} \in \mathbb{R}^{L \times J}$, a value-representation of the input data generated by the transform, $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^{(v)}$, where $\boldsymbol{W}^{(v)} \in \mathbb{R}^{I \times J}$ is a trainable value-weight matrix.

Given $\boldsymbol{K}$, the symmetric attention coefficient matrix, $\boldsymbol{\Theta} \in \mathbb{R}^{L \times L}$, is next computed, where the attention coefficient between the l_1-th and l_2-th time-step is defined as $\boldsymbol{\Theta}_{l_1,l_2} = \epsilon(\frac{1}{\sqrt{d_k}}\boldsymbol{k}_{l_1}^T\boldsymbol{k}_{l_2})$ for $l_1 \neq l_2$ and $\boldsymbol{\Theta}_{l_1,l_2} = 0$ otherwise, with $\epsilon(\cdot)$ as a $relu$ activation function, $\frac{1}{\sqrt{d_k}}$ as a scaling factor, and $\boldsymbol{k}_{l_1}$ and $\boldsymbol{k}_{l_2}$ as respectively the key-vectors at the l_1-th and l_2-th time-step. By viewing each of the L time-steps as nodes of a time-domain graph [11], we can interpret $\boldsymbol{\Theta}$ as an undirected graph adjacency matrix, where the edge weight between the l_1-th and l_2-th vertices, $\boldsymbol{\Theta}_{l_1,l_2}$, is context dependent and generated by a pair-wise similarity function parameterized by $\boldsymbol{W}^{(k)}$, as discussed in Section 2.2.

Remark 25 *The choice of a $relu$ activation to generate the adjacency matrix, $\boldsymbol{\Theta}$, serves two purposes: (i) the rectification promotes sparsity, which effectively reduces the number of connections between the nodes in the graph domain, as the weights are reduced to zero; (ii) the resulting entries are strictly non-negative, which is an usual assumption for weighted graph adjacency matrices.*

Due to its dot product formulation, $\boldsymbol{\Theta}$ is context dependent, but it is ignorant to the time-series structure of the input data, which contains important sequence ordering information. To this end, we incorporate the bi-directional time graph adjacency matrix, $\boldsymbol{\Omega}$, as discussed in Section 10.2.2, to define the adjacency matrix of the time-domain attention graph, $\boldsymbol{\Psi} \in \mathbb{R}^{L \times L}$, as $\boldsymbol{\Psi} = \boldsymbol{\Omega} \odot \boldsymbol{\Theta}$.

Remark 26 *Unlike classical attention, the time-domain attention adjacency matrix, $\boldsymbol{\Psi}$, is both time- and context-aware. Indeed, given a signal $\boldsymbol{x} \in \mathbb{R}^L$, the operation, $\boldsymbol{y} = \boldsymbol{\Psi}\boldsymbol{x}$,*

can be expressed in an element-wise form, $\boldsymbol{y}_{l_1} = \sum_{l_2 \in \mathcal{N}_{l_1}} \boldsymbol{\Theta}_{l_1,l_2} \boldsymbol{\Omega}_{l_1,l_2} x_{l_2}$*. This represents a weighted sum of signals in the neighbourhood of the* l_1*-th time-step,* $\mathcal{N}_{l_1}$*, with weights proportional to both the contextual relevance,* $\boldsymbol{\Theta}_{l_1,l_2}$*, and the time proximity,* $\boldsymbol{\Omega}_{l_1,l_2}$*, between tokens at time-steps* l_1 *and* l_2*.*

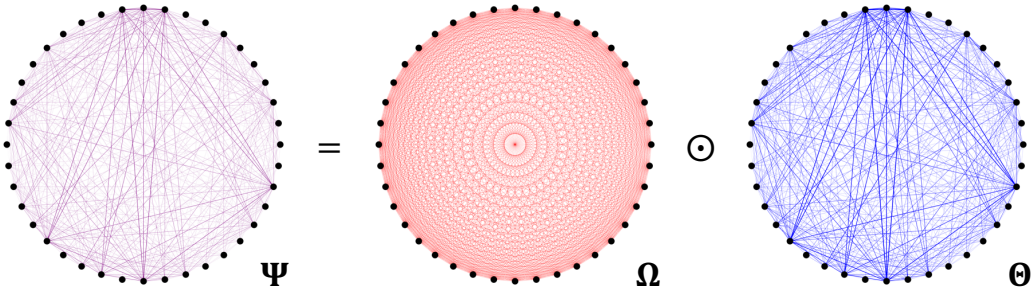


Figure 10.1: Principle of the time-domain attention graph (in purple), computed from the bi-directional time-graph (in red), and the symmetric attention coefficient graph (in blue), as a Hadamard product of their adjacency matrices, $\boldsymbol{\Psi} = \boldsymbol{\Omega} \odot \boldsymbol{\Theta}$. The graph vertices represent a sequence of tokens embedded in the tensor space, which are interconnected with edge weights proportional to their time-distance, $\boldsymbol{\Omega}_{l_1,l_2}$, and attention coefficient, $\boldsymbol{\Theta}_{l_1,l_2}$, for $l_1 = 1, \dots, L$ and $l_2 = 1, \dots, L$.

10.3.2 *Spectral Attention Filtering*

By construction, $\boldsymbol{\Psi}$ is a symmetric graph adjacency matrix representing an undirected graph, which admits an orthonormal eigen-decomposition of $\boldsymbol{\Psi} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^T$, as discussed in Section 2.2. Generally, we can learn a filter, $\boldsymbol{\Gamma}$, in the graph Fourier domain to perform spectral convolution on a graph signal as $\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{\Gamma}\boldsymbol{\Phi}^T\boldsymbol{x}$. To reduce the risk of over-fitting, we can restrict the class of graph filters to be a polynomial function of the eigenvalues of order $(K-1)$ to ensure smooth spectral multipliers [33], that is $\boldsymbol{\Gamma} = f_\alpha(\boldsymbol{\Lambda})$, such that $f_\alpha(\boldsymbol{\Lambda}) = \sum_{k=0}^{K-1} \boldsymbol{\alpha}_k \boldsymbol{\Lambda}^k$, where $\boldsymbol{\alpha}$ is the set of filter coefficients. This simplifies the spectral convolution to

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{\Phi}\boldsymbol{\Gamma}\boldsymbol{\Phi}^T\boldsymbol{x} \\ &= \boldsymbol{\Phi} f_\alpha(\boldsymbol{\Lambda})\boldsymbol{\Phi}^T\boldsymbol{x} \\ &= \boldsymbol{\Phi}(\boldsymbol{\alpha}_0\boldsymbol{\Lambda}^0 + \cdots + \boldsymbol{\alpha}_{\boldsymbol{K-1}}\boldsymbol{\Lambda}^{K-1})\boldsymbol{\Phi}^T\boldsymbol{x} \\ &= (\boldsymbol{\alpha}_0\boldsymbol{\Phi}\boldsymbol{\Lambda}^0\boldsymbol{\Phi}^T + \cdots + \boldsymbol{\alpha}_{K-1}\boldsymbol{\Phi}\boldsymbol{\Lambda}^{K-1}\boldsymbol{\Phi}^T)\boldsymbol{x} \\ &= (\boldsymbol{\alpha}_0\boldsymbol{\Psi}^0 + \cdots + \boldsymbol{\alpha}_{K-1}\boldsymbol{\Psi}^{K-1})\boldsymbol{x} \\ &= f_{\boldsymbol{\alpha}}(\boldsymbol{\Psi})\boldsymbol{x} \end{aligned} \tag{10.1}$$

To reduce the computational and parameter complexity of the proposed attention filtering operation, $\boldsymbol{y} = f_\alpha(\boldsymbol{\Psi})\boldsymbol{x}$, we can restrict the order of the graph filter to $K = 2$, which simplifies (10.1) to $\boldsymbol{y} = (\boldsymbol{\alpha}_0\boldsymbol{I} + \boldsymbol{\alpha}_1\boldsymbol{\Psi})\boldsymbol{x}$. Furthermore, because $\boldsymbol{\Psi}$ is inherently trainable, the coefficient α_1 can be absorbed in $\boldsymbol{\Psi}$ and $\boldsymbol{\alpha}_0$ can be set to 1, which simplifies the attention filtering to $\boldsymbol{y} = (\boldsymbol{I} + \boldsymbol{\Psi})\boldsymbol{x}$.

Remark 27 *Notice that the low complexity filtering operation, $\boldsymbol{y} = (\boldsymbol{I} + \boldsymbol{\Psi})\boldsymbol{x}$, is a special case of the RGTN forward pass discussed in Section 10.2.2, $\boldsymbol{\mathcal{Y}} = (\boldsymbol{I} + \boldsymbol{\Psi}) \times_2^1 \boldsymbol{\mathcal{X}}$, which generalizes the graph filtering operation to process higher-order tensors.*

Remark 28 *The full order-$(K-1)$ filtering operation, $\boldsymbol{y} = f_\alpha(\boldsymbol{\Psi})\boldsymbol{x}$, operates in a $(K-1)$-hop neighbourhood in terms of node (or time-step) connections. The simplified attention filtering, $\boldsymbol{y} = (\boldsymbol{I} + \boldsymbol{\Psi})\boldsymbol{x}$, operates in a 1-hop neighbourhood. However, within the proposed model, we can reach $(K-1)$-hop neighbours by stacking $(K-1)$ layers of simplified attention filtering [37].*

Finally, similar to the classical multi-head attention mechanism, we can repeat the attention filtering operation H times, using a set of attention filters, $\{\boldsymbol{\Psi}_1, \ldots, \boldsymbol{\Psi}_H\}$, to generate H different filtered signals that can be then combined together to generate the final output.

10.4 Tensorizing Attention

10.4.1 *Tensorizing Input Data*

Traditional NLP models process a sequence of tokens (e.g. words) of length, L, embedded in a vector space of dimension, I, resulting in an input matrix, $\boldsymbol{X} \in \mathbb{R}^{L \times I}$. However, due to the "flat-view" nature of matrix methods, performing attention on $\boldsymbol{X}$ incurs expensive matrix multiplications. This results in high complexity costs, especially when working with large embedding dimensions.

To alleviate the complexity costs of the attention mechanism, we embed the sequence of symbols in the tensor space to generate the input tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{L \times I_1 \times \cdots \times I_N}$, and thus benefiting from the ability of GTNs to operate with high-dimensional tensors at low-complexity. By formulating the problem in the tensor space, we can leverage on the power of TNs to drastically reduce the complexity needed to achieve high expressive power.

10.4.2 *Tensorizing Large Dimensional Matrix Multiplications*

Consider a large-dimensional matrix multiplication of the form, $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W}$, where the matrices are of dimensionality $\boldsymbol{Y} \in \mathbb{R}^{L \times J}$, $\boldsymbol{X} \in \mathbb{R}^{L \times I}$, and $\boldsymbol{W} \in \mathbb{R}^{I \times J}$. Without loss of generality, let the dimensionality of the considered matrices be factorizable as $I = I_1 \times \cdots \times I_N$ and $J = J_1 \times \cdots \times J_N$. This allows us to tensorize $\boldsymbol{X}$ to $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{L \times I_1 \times \cdots \times I_N}$, $\boldsymbol{Y}$ to $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{L \times J_1 \times \cdots \times J_N}$, and $\boldsymbol{W}$ to $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{I_1 \times J_1 \times \cdots \times I_N \times J_N}$. In turn, this allows us to re-write the large-dimensional matrix multiplication, which is ubiquitous to the attention mechanism, as a higher order tensor contraction

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_{2,3,4,\cdots,N+1}^{1,3,5,\cdots,2N-1} \boldsymbol{\mathcal{W}} \tag{10.2}$$

By virtue of the above tensor structure, we can now apply TTD to store the weight tensor, $\boldsymbol{\mathcal{W}}$, in a low-rank Matrix-Product-Operator (MPO) form

$$\boldsymbol{\mathcal{W}} \approx \boldsymbol{\mathcal{G}}_1 \times_4^1 \cdots \times_4^1 \boldsymbol{\mathcal{G}}_N \tag{10.3}$$

where $\boldsymbol{\mathcal{G}}_n \in \mathbb{R}^{R_{n-1} \times I_n \times J_n \times R_n}$ for $n = 1, \ldots, N$ are referred to as the TT-cores, and the set $\{R_0, R_1, \ldots, R_N\}$ is the TT-rank of the TT decomposition, where $R_0 = R_N = 1$.

Remark 29 *In the attention mechanism context, instead of learning a large dimensional weight matrix, $\boldsymbol{\mathcal{W}}$, we can learn directly the low-rank TT-cores, $\boldsymbol{\mathcal{G}}_n$ for $n = 1, \ldots, N$, which reduces the complexity from an exponential $\mathcal{O}\big(\prod_{n=1}^{N} I_n J_n\big) = \mathcal{O}\big(IJ\big)$, to a linear $\mathcal{O}\big(\sum_{n=1}^{N} R_{n-1} I_n J_n R_n\big)$, in terms of the dimensions I_n and J_n. This is highly efficient for low TT-rank and large number of dimensions N.*

By storing $\boldsymbol{\mathcal{W}}$ in the TT format, we can express the large-scale contraction in (10.2), as a series of smaller-scale tensor contractions

$$\begin{aligned} \boldsymbol{\mathcal{Y}} &= \boldsymbol{\mathcal{X}} \times_{2,3,4,\cdots,N+1}^{1,3,5,\cdots,2N-1} \Big(\boldsymbol{\mathcal{W}}\Big) \\ &= \boldsymbol{\mathcal{X}} \times_{2,3,4,\cdots,N+1}^{1,3,5,\cdots,2N-1} \Big(\boldsymbol{\mathcal{G}}_1 \times_4^1 \boldsymbol{\mathcal{G}}_2 \times_4^1 \boldsymbol{\mathcal{G}}_3 \times_4^1 \cdots \times_4^1 \boldsymbol{\mathcal{G}}_N\Big) \end{aligned} \tag{10.4}$$

To further reduce the parameter complexity, we can employ the notion of QTTD, by: (i) setting the dimensions I and J to be a power of 2, such that $I_n = J_n = 2$ for $n = 1, \ldots, N$, and (ii) setting the TT-rank $R_n = 2$ for $n = 1, \ldots, N-1$. This effectively *super-compresses* the parameter complexity from an exponential $\prod_{n=1}^{N} I_n J_n = 4^N$ to a linear $\sum_{n=1}^{N} R_{n-1} I_n J_n R_n = 16(N-1)$.

Remark 30 *The use of the compressed TT format in (10.4) to replace all dense matrix multiplications in the attention mechanism makes it possible to drastically reduce the parameter complexity from an exponential one to a linear one in the tensor order N.*

10.4.3 *Tensorizing Attention Coefficients*

To perform spectral attention filtering in the tensor space, we need to compute the attention coefficient matrix, $\boldsymbol{\Theta}$, from the tensorized input. To achieve this, we can first compute the key-representation of the input data, $\boldsymbol{\mathcal{K}} \in \mathbb{R}^{L \times J_1 \times \cdots \times J_N}$, by employing (10.4), then compute the coefficients, $\boldsymbol{\Theta}_{l_1,l_2}$, as $\boldsymbol{\Theta}_{l_1,l_2} = \epsilon(\frac{1}{\sqrt{d_k}} \langle \boldsymbol{\mathcal{K}}_{l_1}, \boldsymbol{\mathcal{K}}_{l_2} \rangle)$, where $\boldsymbol{\mathcal{K}}_{l_1} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$ and $\boldsymbol{\mathcal{K}}_{l_2} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$ are the slices of $\boldsymbol{\mathcal{K}}$ at the l_1-th and l_2-th time-steps. This is also equivalent to computing $\boldsymbol{\Theta} = \epsilon(\frac{1}{\sqrt{d_k}} \boldsymbol{\mathcal{K}} \times_{2,\cdots,N+1}^{2,\cdots,N+1} \boldsymbol{\mathcal{K}})$ and then setting its diagonal elements to zero.

10.5 Putting Everything Together

The GTN based TSA is summarised in Algorithm 1, which can be repeated for H different heads to perform multi-head TSA. In practice, the condition $I_n = J_n = 2$ is set for Algorithm 1 to *super-compress* the parameter complexity costs.

Algorithm 6: TSA Algorithm

Input : Data matrix $\boldsymbol{X}$ and time-domain adjacency matrix $\boldsymbol{\Omega}$
Output : Feature matrix $\boldsymbol{Y}$
Initialize : Initialize $R_n = 2$ for $n = 1, \ldots, N-1$ and $R_0 = R_N = 1$
Initialize $I_1, \ldots, I_N$ such that $\prod_{n=1}^{N} I_n = I$
Initialize $J_1, \ldots, J_N$ such that $\prod_{n=1}^{N} J_n = J$
Initialize trainable key TT-cores $\boldsymbol{\mathcal{G}}_n^{(k)}$
Initialize trainable value TT-cores $\boldsymbol{\mathcal{G}}_n^{(v)}$

$\boldsymbol{\mathcal{X}} \leftarrow \text{ten}(\boldsymbol{X})$ // Tensorization from $\boldsymbol{X} \in \mathbb{R}^{L \times I}$ to $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{L \times I_1 \times \cdots \times I_N}$
$\boldsymbol{\mathcal{K}} \leftarrow \boldsymbol{\mathcal{X}} \times_2^2 \boldsymbol{\mathcal{G}}_1^{(k)} \times_{2,N+3}^{2,1} \boldsymbol{\mathcal{G}}_2^{(k)} \times_{2,N+3}^{2,1} \cdots \times_{2,N+3}^{2,1} \boldsymbol{\mathcal{G}}_N^{(k)}$ // Key tensor
$\boldsymbol{\mathcal{V}} \leftarrow \boldsymbol{\mathcal{X}} \times_2^2 \boldsymbol{\mathcal{G}}_1^{(v)} \times_{2,N+3}^{2,1} \boldsymbol{\mathcal{G}}_2^{(v)} \times_{2,N+3}^{2,1} \cdots \times_{2,N+3}^{2,1} \boldsymbol{\mathcal{G}}_N^{(v)}$ // Value tensor
$\boldsymbol{\Theta} \leftarrow \epsilon(\frac{1}{\sqrt{d_k}} \boldsymbol{\mathcal{K}} \times_{2,\cdots,N+1}^{2,\cdots,N+1} \boldsymbol{\mathcal{K}})$ // Attention coefficient matrix
$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \text{diag}(\boldsymbol{\Theta})$ // Remove diagonal
$\boldsymbol{\Psi} \leftarrow \boldsymbol{\Omega} \odot \boldsymbol{\Theta}$ // Time-domain attention adjacency matrix
$\boldsymbol{\mathcal{Y}} \leftarrow (\boldsymbol{I} + \boldsymbol{\Psi}) \times_2^1 \boldsymbol{\mathcal{V}}$ // Spectral attention filtering of the tensor embeddings
$\boldsymbol{Y} \leftarrow \text{mat}(\boldsymbol{\mathcal{Y}})$ // Matricization

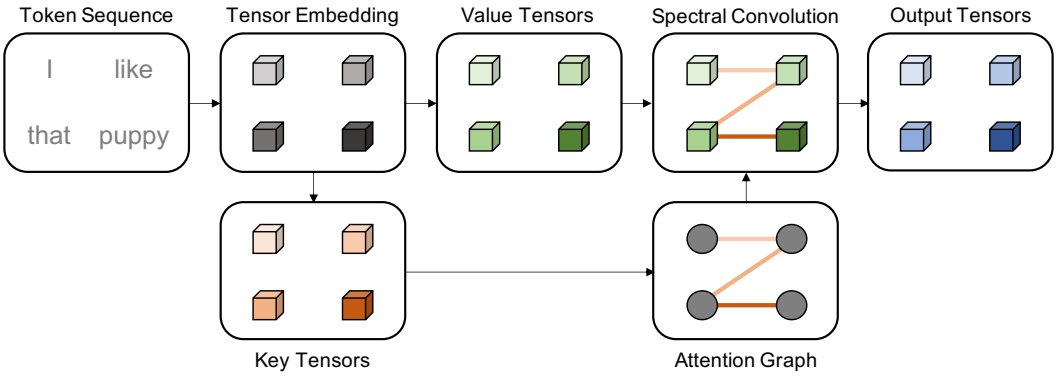


Figure 10.2: Principle of the TSA mechanism. A sequence of tokens are embedded in the tensor space (in gray). The sequence of tensor embeddings are then mapped to generate the key tensors, $\boldsymbol{\mathcal{K}}$ (in orange), and the value tensors, $\boldsymbol{\mathcal{V}}$ (in green). The key tensors are then used to generate the adjacency matrix of the attention graph, $\boldsymbol{\Psi}$, which is used to perform spectral graph convolution on value tensors to generate the output tensors, $\boldsymbol{\mathcal{Y}}$ (in blue).

10.6 Experimental Results

The performance of the proposed TSA was evaluated in terms of its parameter complexity, expressive power, and performance on various NLP tasks. Experimental results confirm the superiority of the proposed model against standard attention mechanisms, achieving a better or comparable performance at a drastically lower complexity across five different datasets.

Experiment Setting: To benchmark the performance of the proposed TSA against classical attention mechanisms, we compared the training and testing accuracy of the considered models at different complexity levels across five NLP benchmark datasets. More specifically, we compared the achieved accuracy scores by varying the word embedding dimension in the attention layer while keeping all other parameters constant, which resulted in models of varying expressive power and complexity. This allowed us to analyse the performance-complexity trade-offs of the considered models. Finally, the experiments were repeated over 5 independent trials for each benchmark dataset.

Metrics: To evaluate the performance of the proposed model, we considered: (i) Training Accuracy (TrA) as a measure of expressive power, (ii) Testing Accuracy (TeA) as a measure of performance, and (iii) Number of Parameters (Params) as a measure of model complexity.

Datasets: Five benchmark NLP classification datasets were considered for the evaluation of the TSA model: (i) Reuters Newswire[1] (RN), (ii) 20 Newsgroups[2] (NG), (iii) Drugs Review[3] (DR), (iv) Research Articles[4] (RA), and (v) Spooky Authors Identification[5] (SA). All datasets consist of text data that were pre-processed as tokenized sequences of 200 words by considering the 20,000 most common words in the dataset. A train-test split of 60%-40% was used for all datasets.

Baselines: We benchmarked the proposed TSA against Luong-style DPA and Bahdanau-style ADditive Attention (ADA) based models. For a fair comparison, all considered models have the exact same training setting and model architecture, with the sole exception being the attention mechanism used. More specifically, we implemented a seven-layer architecture consisting of: (i) word embedding layer, (ii) multi-head ($H = 2$) attention layer (based either on the TSA, DPA, or ADA), (iii) global average pooling layer, (iv) drop-out layer with 10% drop-out rate, (v) dense layer with 20 hidden units and $relu$ activation, (vi) drop-out layer with 10% drop-out rate, and (vii) dense layer with $Softmax$ activation. The models were trained using the categorical cross-entropy loss function with an *Adam* optimizer.

[1]Available at https://trec.nist.gov/data/reuters/reuters.html
[2]Available at http://people.csail.mit.edu/jrennie/20Newsgroups
[3]Available at https://archive.ics.uci.edu/ml/datasets/Drug+Review+Dataset+%28Drugs.com%29
[4]Available at https://www.kaggle.com/blessondensil294/topic-modeling-for-research-articles
[5]Available at https://www.kaggle.com/c/spooky-author-identification

Remark 31 *As discussed in Section 10.1, we did not compare the proposed TSA against the TensorCoder in [154] due to its dimension-wise attention formulation, and the Multi-Linear Attention in [152] due to its constraint on the embedding dimension, which has to be at least equal to the sequence length for a fair comparison.*

Implementation: All models were implemented using TensorFlow 2.4.0 and trained on a 2.4 GHz 8-Core Intel Core i9 CPU with 32GB RAM. The experiments took approximately 2 days to complete. For additional details, please refer to the full code at the provided link.

Results: The experiment results are illustrated in Figure 10.3. Across varying levels of word embedding dimensionality, the proposed TSA is shown to achieve better or comparable expressive power (measured in training accuracy) and performance (measured in testing accuracy), while incurring drastically lower complexity costs (measured in the number of parameters). Table 10.1 summarises the results at a low complexity level of around 300 trainable parameters, which further highlights the strong performance-to-complexity characteristics of the proposed TSA, as it achieved remarkably better performances compared to standard attention mechanisms.

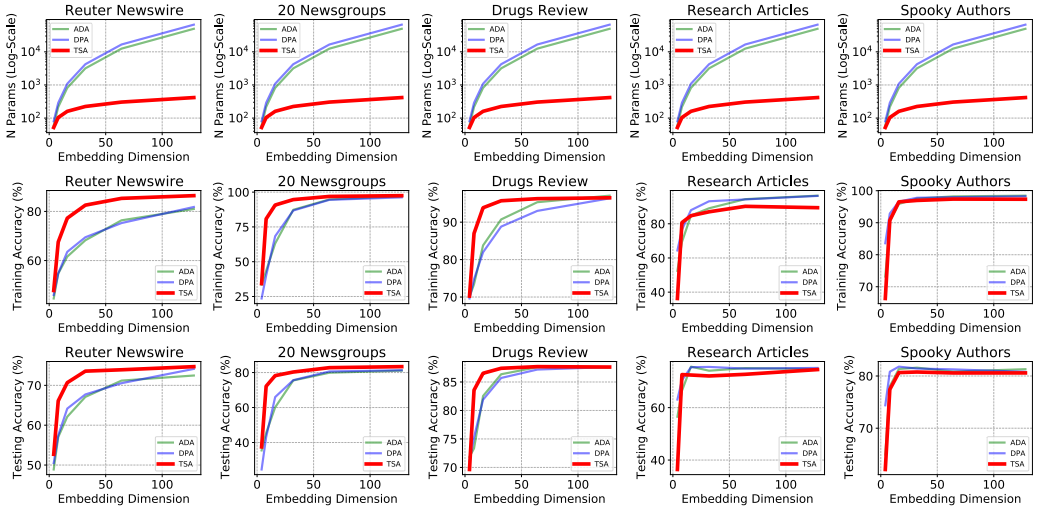


Figure 10.3: Performance comparison of the proposed TSA, DPA, and ADA models. The figure columns, from left to right, correspond to the five considered datasets. The figure rows, from top to bottom, correspond to: (i) embedding dimension vs number of parameters (complexity), (ii) embedding dimension vs training accuracy (expressive power), and (iii) embedding dimension vs testing accuracy (performance).

Remark 32 *As shown in Table 1, at a low complexity level (using around 300 trainable parameters), the proposed TSA framework achieved substantially better results than the classical attention model, which generated poor performance due to the excessively small embedding dimension ($I = 6$) needed to satisfy the low complexity requirement.*

Metric	Model	RN	NG	DR	RA	SA
Training Acc. (%)	ADA	57.5 ± 2.7	48.7 ± 12.9	77.0 ± 1.9	76.0 ± 4.9	93.9 ± 1.2
	DPA	55.7 ± 4.4	46.0 ± 10.0	74.7 ± 4.3	80.3 ± 2.4	92.6 ± 2.3
	TSA	$\mathbf{85.3 \pm 1.0}$	$\mathbf{96.9 \pm 0.5}$	$\mathbf{96.3 \pm 0.2}$	$\mathbf{90.2 \pm 3.0}$	$\mathbf{97.4 \pm 0.4}$
Test Acc. (%)	ADA	58.4 ± 3.0	49.0 ± 13.1	77.6 ± 2.8	71.9 ± 3.2	$\mathbf{81.3 \pm 0.7}$
	DPA	58.2 ± 3.6	49.0 ± 9.0	75.7 ± 4.5	71.8 ± 2.4	81.3 ± 1.6
	TSA	$\mathbf{73.9 \pm 0.5}$	$\mathbf{82.8 \pm 1.0}$	$\mathbf{87.7 \pm 0.4}$	$\mathbf{72.7 \pm 0.9}$	80.6 ± 0.8

Table 10.1: Expressive power measured in training accuracy (top table), and performance measured in testing accuracy (bottom table) of the considered models at a low complexity level (measured in number of parameters of around 300) across five datasets (RN, NG, DR, RA, SA).

10.7 Summary

We have introduced a novel TSA mechanism for NLP. By virtue of its conjoint tensor and graph formulation, the proposed framework has been shown to achieve high expressive power at a drastically lower parameter complexity compared to traditional attention models. Experimental results have verified the desirable properties of the proposed TSA, with better or comparable performance achieved across five NLP benchmark tasks at varying levels of complexity. The performance gain has been demonstrated to be especially fruitful in low complexity cases.

Chapter 11

Generalized Graph Tensor Networks

Despite the omnipresence of tensors and tensor operations in modern deep learning, the use of tensor algebra to formally design and describe neural networks is still under-explored within the deep learning community. To this end, we introduce the GTN framework, an intuitive yet rigorous graphical framework for systematically designing and implementing large-scale neural learning systems on both regular and irregular domains. This is achieved by extending the results from the previous two chapters to a more general setting through the Tucker product. The proposed framework is shown to be general enough to include many popular architectures as special cases, and flexible enough to handle data on any and many data domains. This opens avenues for the design of new and efficient neural networks which are not possible to arrive at otherwise. The power and flexibility of the proposed framework is demonstrated through real-data experiments.

11.1 Introduction

Modern neural networks are effectively comprised of a cascade of tensor operations that are interleaved with non-linear activation functions. However, despite the omnipresence of tensors in deep learning research and popular software libraries, the use of tensor mathematics to formally describe and design neural learning systems is not yet common practice within the research community.

Tensors are a multi-dimensional generalization of vectors (order-1 tensors) and matrices (order-2 tensors). By virtue of their multi-linear algebra, tensor operations exploit the multi-dimensional structure inherent to big data applications to efficiently operate on large-dimensional data at low complexity, while preserving their inherent structure and interpretability [20]. In this way, tensors help bypass the bottlenecks imposed by the curse of dimensionality, reducing the associated computational costs from an exponential one to a linear one in the data dimensions [18, 19]. Despite these advantages, the multi-linear algebra, a branch of mathematics dealing with tensors, is not as intuitive as the classical linear algebra. In addition, tensor expressions are not as easily manipulated as their matrix counterparts, and the notation can be overwhelming due to the necessity of indexing operations across multiple dimensions. As a remedy to these conceptual obstacles, graphical approaches known as TNs

have been developed in the quantum physics community, with the aim to illustrate and implement complex tensor operations as mathematical graphs, in an insightful and mathematically rigorous manner [64, 65, 2]. The design of neural networks would therefore greatly benefit from the TN framework, as it would allow for the visualization and manipulation of large-scale, multi-dimensional operations through simple graphical illustrations, thus bypassing the cumbersome multi-index operations associated with tensor manipulation.

To this end, we introduce a TN based framework for describing and designing neural networks through tensor mathematics. We show that this enables an intuitive and rigorous way of designing neural architectures for large- and multi-dimensional data across any and many data domains. This is achieved by a specific type of TN which computes the Tucker product of the data tensor with appropriately designed GSO matrices, which incorporates knowledge about domain-specific structures present within the data tensor. The tensor contraction between the data tensor and the GSO matrices then effectively performs joint modelling of data across multiple data domains, which can be naturally represented as a TN, resulting in the proposed GTN framework. The GTN framework is shown to be general enough to include many modern neural network architectures as special cases, where the primary difference in the respective architectures is captured by the design of the GSO. This unifying framework is different from most existing work, as the difference in neural network architecture arises from different designs of GSOs within the same GTN framework, instead of considering different TNs. Furthermore, the proposed framework opens up avenues for the inclusion of more sophisticated tensor techniques, such as those developed in quantum physics, in order to design new classes of efficient neural networks, which are not possible to arrive at otherwise.

The rest of the chapter is organised as follows. Related work is first discussed in Section 11.2. The preliminaries necessary to follow this work are presented in Section 11.3. The multi-domain data tensor is defined in Section 11.4. The proposed GTN framework is introduced in Section 11.5. Several classical neural network architectures, such as Dense, Convolutional, Graph, Attention, and RNNs, are shown to be special cases of the proposed framework, as elaborated in Section 11.6. Section 11.7 discusses the application of the considered framework to multi-domain modelling. The power and flexibility of the proposed framework is then demonstrated through real-data experiments in Section 11.8. Finally, Section 11.9 concludes the approach and summarizes the findings.

11.2 Related Work

The joint consideration of graphs, tensors, and neural networks offers enormous potential and was first discussed in [11, 16]. This was achieved by leveraging on the concept of graph filters and TNs to develop low-complexity neural networks for time-domain modelling. Despite success, this initial work was highly specialized, as the motivation arose from the application domain rather than aiming to solve a general learning paradigm. The present work instead builds on these initial results by extending the considered framework to handle tensor-variate data on any and

many domains, which is general enough to include numerous existing neural network architectures as special cases. The use of TD techniques to reduce the complexity of neural networks was explored for many architectures, such as DNNs [28, 74], CNNs [158], RNNs [29, 13], and the Attention Mechanism [10]. However, the existing work was primarily concerned with the compression of neural networks, while this work introduces a general framework for describing neural networks under one unified umbrella of TNs. Authors in [30] were the first to analyze the expressive power of CNNs through TNs. However, instead of considering only one particular architecture, the framework proposed here applies more generally across many existing architectures, together with providing a platform for the design of new neural networks, which are not possible to arrive at otherwise. The idea of multi-graph modelling via neural networks was also discussed in [159, 160], however, these results did not leverage on the graphical method of TNs. The proposed framework is also different from graph-regularized tensor models [123, 125, 124], where the graph is used to regularize the TD instead of processing data on graphs. Other prominent work discussing the use of tensors and TNs to solve ML problems include [161, 23, 162, 163] for regression tasks and [126, 164, 165] for classification tasks. However, the focus of these works was on shallow learning algorithms based on specific types of TD, while we here consider a more general design framework that can be extended to deep learning architectures.

11.3 Preliminaries

Convolution Tensor: The Convolution tensor, $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{I \times I \times P}$, where $I > P$, is a sparse order-3 tensor with entries $\boldsymbol{\mathcal{C}}_{i,(i+p-1)\%I,p} = 1$ for $i = 1, \ldots, I$ and $p = 1, \ldots, P$, where $\%$ denotes the modulo operation. Given an input vector, $\boldsymbol{x} \in \mathbb{R}^I$, and a convolution kernel, $\boldsymbol{k} \in \mathbb{R}^P$, the discrete convolution can be obtained through the tensor contraction $\boldsymbol{y} = \boldsymbol{x} \circledast \boldsymbol{k} = \boldsymbol{\mathcal{C}} \times_2^1 \boldsymbol{x} \times_3^1 \boldsymbol{k} = \boldsymbol{S}\boldsymbol{x} \in \mathbb{R}^L$, where $\boldsymbol{S} = \boldsymbol{\mathcal{C}} \times_3^1 \boldsymbol{k} \in \mathbb{R}^{I \times I}$ is a circulant matrix generated from the convolution kernel, $\boldsymbol{k}$. The circulant matrix, $\boldsymbol{S}$, can be interpreted as the adjacency matrix of a circulant graph, which effectively casts the standard convolution as a special case of graph convolution on a circulant graph. The convolution tensor has an exact QTTD low-rank format.

11.4 Multi-Domain Data Tensors:

Modern data sources often reside on multiple domains which naturally admit a tensor representation. More specifically, data collected over N domains will naturally result in a tensor with N domain modes, where each mode corresponds to a physically meaningful data domain which can be described as a graph. Formally, a multi-domain data tensor is defined as:

Data Tensor: An order-$(N+M)$ data tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N \times J_1 \times \cdots \times J_M}$, is a collection of input data over N domain modes and M feature modes. A domain mode of size I_n is a mode with a well-defined structure that can be described as a graph (e.g., the

image pixels arranged as a regular grid graph), while a feature mode of size J_m is a mode with a collection of descriptive features (e.g., the RGB value of image pixels). Such a data tensor is general enough to describe the inputs to most neural network architectures, as discussed in Section 11.6.

Example 8 *A collection of sensor measurements forms a feature vector, which is an order-*1 *data tensor with* $M = 1$ *feature mode. If such measurements are collected over multiple sensors in a traffic network, then the input data forms an order-*2 *data tensor, which is a matrix with* $N = 1$ *domain mode representing sensors and* $M = 1$ *feature mode. If such data is also collected over time, then the input data forms an order-*3 *tensor, with* $N = 2$ *domain modes representing the sensors-domain and time-domain, and* $M = 1$ *feature mode. The physical placement of sensors can be naturally described as a graph, where the nodes represent sensors and the edges encode their physical proximity. Similarly, the time-domain can also be described as a graph, where the nodes represent time-steps, and the edges encode the directed flow of information over time (from past to future).*

In practice, the design of neural networks to process data will be highly dependent on the underlying data domains. For instance, the time-domain can be processed by a RNN, while the sensors domain can be processed by a GNN (GNN). However, these classical neural network architectures were designed using "flat-view" matrix methods, which are restricted to handle only one data domain at a time. This renders the visualisation and manipulation of data in multiple domains awkward and difficult to implement. Unlike matrix-based methods, by taking advantage of the inherent tensor structure naturally present in multi-domain data, the proposed GTN framework is capable of jointly processing multiple data domains via multi-way tensor contractions with appropriately designed GSOs, which is both intuitive and efficient under the TN framework.

11.5 The Graph Tensor Network Framework

Given the ability of graphs to operate on both irregular and regular domains, the ability of tensors to manipulate data across several dimensions, and the universal function approximation property of neural networks, it is therefore natural to investigate their conjoint treatment. To this end, we introduce the GTN framework, which is suitable for both describing and designing neural networks that handle large-dimensional data on both regular and irregular domains. The GTN framework is next shown to be general enough to include most modern neural network architectures as special cases. By virtue of graphs and tensors, the proposed GTN framework is shown to be flexible enough to handle tensor-variate data on any and many domains. In addition, through the use of TN diagrams, the proposed framework allows for the design of large-scale neural learning systems in a highly intuitive and mathematically rigorous manner.

11.5.1 *The Forward Pass*

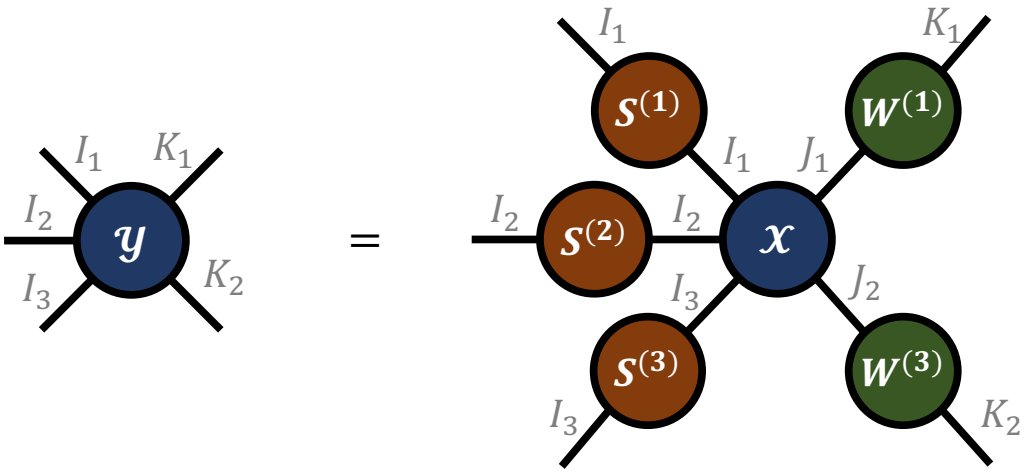


Figure 11.1: The forward pass of a GTN for an order-$(3+2)$ input data tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times J_1 \times J_2}$, according to (11.1).

Consider a ML paradigm where the input tensor is an order-$(N+M)$ data tensor with N domain modes and M feature modes, as defined in Section 11.4, $\boldsymbol{\mathcal{X}}$. The proposed GTN framework performs a Tucker product forward pass, given by

$$\boldsymbol{\mathcal{Y}} = [\![\boldsymbol{\mathcal{X}}; \boldsymbol{S}^{(1)}, \ldots, \boldsymbol{S}^{(N)}, \boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(M)}]\!] \tag{11.1}$$

where $\boldsymbol{S}^{(n)} \in \mathbb{R}^{I_n \times I_n}$ is the mode-n GSO applied to the n-th domain mode (there are as many graph filters as the number of domain modes) and $\boldsymbol{W}^{(m)} \in \mathbb{R}^{K_m \times J_m}$ is the mode-m weight matrix applied to the m-th feature mode (there are as many weight matrices as the number of feature modes). Figure 11.1 illustrates the forward pass in (11.1) in the TN notation. Similar to classical neural networks, a bias tensor, $\boldsymbol{\mathcal{B}}$, can be optionally added to the final transform, followed by an activation function to introduce non-linearity.

Remark 33 *The dimensions of the tensor operations are unchanged by the application of an activation function due to its element-wise nature. This implies that the TN topology is independent of the activation function.*

Remark 34 *The mode-n products between the input tensor, $\boldsymbol{\mathcal{X}}$, and N GSOs, $\boldsymbol{S}^{(n)} \in \mathbb{R}^{I_n \times I_n}$, can be interpreted as N simultaneous graph shift operations over all N domain modes of the input tensor. Indeed, each mode-n product, $\boldsymbol{\mathcal{X}} \times_n^2 \boldsymbol{S}^{(n)}$, can be interpreted as a mode-n graph operation over the graph signal matrix, $\boldsymbol{X}_{\{n\}}$, as $\boldsymbol{Y} = \boldsymbol{S}^{(n)} \boldsymbol{X}_{\{n\}}$, which follows from the property discussed in Section 2.1.*

11.6 Classical Neural Networks as GTNs

This section derives several neural network architectures as special cases of the GTN forward pass in (11.1), whereby the difference in architectures is primarily accounted for through the design of the corresponding GSO for the domain mode.

Remark 35 *For conciseness, the derivations in this section will omit the bias term and the activation function. However, these can always be added to the final transform without any loss of generality.*

11.6.1 *Dense Neural Network*

A DNN takes as input an order-$(0+1)$ data tensor as defined in Section 11.4 (a feature vector with $N = 0$ domain mode and $M = 1$ feature mode), $\boldsymbol{x} \in \mathbb{R}^{J_1}$, and generates an output vector according to the transform

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} \in \mathbb{R}^{K_1} \tag{11.2}$$

where $\boldsymbol{W} \in \mathbb{R}^{K_1 \times J_1}$ is a trainable weight matrix. The DNN forward pass in (11.2) can be shown to be a special case of (11.1) with $N = 0$ domain modes and $M = 1$ feature mode, as

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{W}\boldsymbol{x} \\ &= \boldsymbol{x} \times_1^2 \boldsymbol{W} \\ &= [\![\boldsymbol{x}; \boldsymbol{W}]\!] \\ &= [\![\boldsymbol{x}; \boldsymbol{W}^{(1)}]\!] \end{aligned} \tag{11.3}$$

where $\boldsymbol{W}^{(1)} = \boldsymbol{W}$ for simplicity of notation. Figure 11.2 illustrates (11.3) in the TN notation.



Figure 11.2: DNN as a GTN with $N = 0$ domain modes and $M = 1$ feature mode, according to (11.3).

11.6.2 *Graph Convolutional Network*

Given the adjacency matrix of a graph, $\boldsymbol{A}$, a GCN [58] takes as input an order-$(1+1)$ data tensor (a graph signal matrix with $N = 1$ graph domain mode and $M = 1$ feature mode), $\boldsymbol{X} \in \mathbb{R}^{I_1 \times J_1}$, and generates an output matrix according to the transform

$$\boldsymbol{Y} = \boldsymbol{S}\boldsymbol{X}\boldsymbol{W} \in \mathbb{R}^{I_1 \times K_1} \tag{11.4}$$

where $\boldsymbol{W}$ is a trainable weight matrix and $\boldsymbol{S} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ is a graph convolution operator with $\tilde{\boldsymbol{A}} = \boldsymbol{I} + \boldsymbol{A} \in \mathbb{R}^{I_1 \times I_1}$ and $\tilde{\boldsymbol{D}} = \text{degree}\left(\tilde{\boldsymbol{A}}\right)$. The GCN forward pass in (11.4) can be shown to be a special case of (11.1) with $N = 1$ domain mode (representing graph vertices) and $M = 1$ feature mode, as

$$\begin{aligned} \boldsymbol{Y} &= \boldsymbol{S}\boldsymbol{X}\boldsymbol{W} \\ &= \boldsymbol{X} \times_1^2 \boldsymbol{S} \times_2^2 \boldsymbol{W}^T \\ &= [\![\boldsymbol{X}; \boldsymbol{S}, \boldsymbol{W}^T]\!] \\ &= [\![\boldsymbol{X}; \boldsymbol{S}^{(1)}, \boldsymbol{W}^{(1)}]\!] \end{aligned} \tag{11.5}$$

where $\boldsymbol{S} = \boldsymbol{S}^{(1)}$ and $\boldsymbol{W}^{(1)} = \boldsymbol{W}^T$ for simplicity of notation. Figure 11.3 illustrates (11.5) in the TN notation.

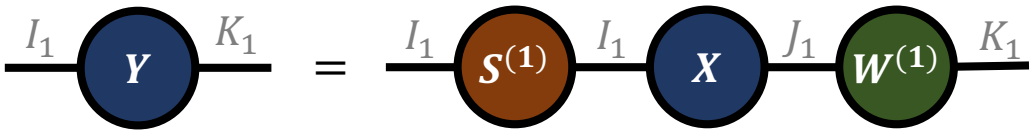


Figure 11.3: GCN as a GTN with $N = 1$ domain mode (representing graph vertices) and $M = 1$ feature mode, according to (11.5).

11.6.3 *Convolutional Neural Network*

A CNN takes as input an order-$(1 + 0)$ data tensor (where the $N = 1$ domain mode corresponds to the convolution domain), $\boldsymbol{x} \in \mathbb{R}^{I_1}$, and generates the output vector through the discrete convolution

$$\boldsymbol{y} = \boldsymbol{x} \star \boldsymbol{k} \tag{11.6}$$

where $\boldsymbol{k}$ is a trainable convolution kernel. By applying the convolution tensor defined in Section 11.3, $\boldsymbol{\mathcal{C}}$, the CNN forward pass in (11.6) can be shown to be a special case of (11.1) with $N = 1$ domain mode (representing the convolution domain) and $M = 0$ feature mode as

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{x} \star \boldsymbol{k} \\ &= \boldsymbol{\mathcal{C}} \times_2^1 \boldsymbol{x} \times_3^1 \boldsymbol{k} \\ &= \left(\boldsymbol{\mathcal{C}} \times_3^1 \boldsymbol{k}\right) \boldsymbol{x} \\ &= \boldsymbol{S}^{(1)}\boldsymbol{x} \\ &= \boldsymbol{x} \times_1^2 \boldsymbol{S}^{(1)} \\ &= [\![\boldsymbol{x}; \boldsymbol{S}^{(1)}]\!] \end{aligned} \tag{11.7}$$

where $\boldsymbol{S}^{(1)} = \boldsymbol{\mathcal{C}} \times_3^1 \boldsymbol{k}$. Figure 11.4 illustrates (11.7) in the TN notation.

Remark 36 *The GSO, $\boldsymbol{S}^{(1)} = \boldsymbol{\mathcal{C}} \times_3^1 \boldsymbol{k}$, is a circulant matrix constructed from the trainable convolution kernel, $\boldsymbol{k} \in \mathbb{R}^P$, which effectively acts as the adjacency matrix of a circulant graph, as discussed in Section 11.3.*

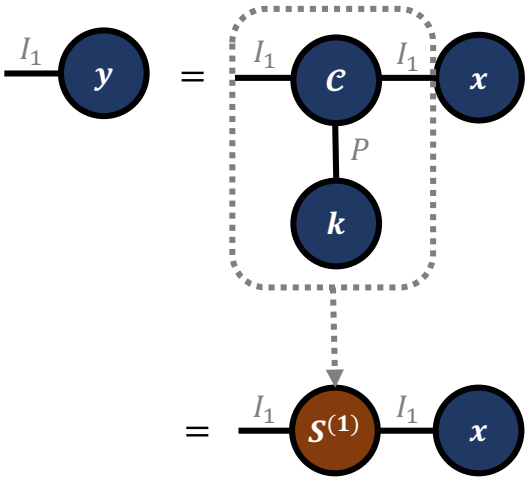


Figure 11.4: CNN as a GTN with $N = 1$ domain mode and $M = 0$ feature mode, according to (11.7).

11.6.4 *Attention Neural Network*

A dot product style Attention Neural Network (ANN) [148], takes as input an order-$(1+1)$ data tensor (a time-series features matrix with $N = 1$ time-domain mode and $M = 1$ feature mode), $\boldsymbol{X} \in \mathbb{R}^{I_1 \times J_1}$, and generates the output matrix according to the transform

$$\boldsymbol{Y} = \sigma\left(\frac{1}{\sqrt{d_k}}\left(\boldsymbol{X}\boldsymbol{W}^{(q)}\right)\left(\boldsymbol{X}\boldsymbol{W}^{(k)}\right)^T\right)\boldsymbol{X}\boldsymbol{W}^{(v)} \tag{11.8}$$

where $\boldsymbol{W}^{(k)} \in \mathbb{R}^{J_1 \times K_1}$, $\boldsymbol{W}^{(q)} \in \mathbb{R}^{J_1 \times K_1}$, and $\boldsymbol{W}^{(v)} \in \mathbb{R}^{J_1 \times K_1}$ are trainable weight matrices, $\sigma(\cdot)$ is a $SoftMax$ activation function, and $\sqrt{d_k}$ is a scaling factor. The attention operation can also be written in a short-hand form as $\boldsymbol{Y} = \sigma\left(\frac{1}{\sqrt{d_k}}\boldsymbol{Q}\boldsymbol{K}^T\right)\boldsymbol{V}$, where $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^{(q)} \in \mathbb{R}^{I_1 \times K_1}$, $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{(k)} \in \mathbb{R}^{I_1 \times K_1}$, and $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^{(v)} \in \mathbb{R}^{I_1 \times K_1}$ are referred to respectively as the Key, Query, and Value matrices. The ANN forward pass in (11.8) can be shown to be a special case of (11.1) with $N = 1$ domain mode (representing time) and $M = 1$ feature mode, to yield

$$\begin{aligned}
\boldsymbol{Y} &= \sigma\left(\frac{1}{\sqrt{d_k}}\left(\boldsymbol{X}\boldsymbol{W}^{(q)}\right)\left(\boldsymbol{X}\boldsymbol{W}^{(k)}\right)^T\right)\boldsymbol{X}\boldsymbol{W}^{(v)} \\
&= \sigma\left(\frac{1}{\sqrt{d_k}}\boldsymbol{Q}\boldsymbol{K}^T\right)\boldsymbol{X}\boldsymbol{W}^{(v)} \\
&= \boldsymbol{S}^{(1)}\boldsymbol{X}\boldsymbol{W}^{(v)} \\
&= \boldsymbol{X} \times_1^2 \boldsymbol{S}^{(1)} \times_2^2 \boldsymbol{W}^{(v)} \\
&= [\![\boldsymbol{X}; \boldsymbol{S}^{(1)}, \boldsymbol{W}^{(v)}]\!] \\
&= [\![\boldsymbol{X}; \boldsymbol{S}^{(1)}, \boldsymbol{W}^{(1)}]\!]
\end{aligned} \tag{11.9}$$

where $\boldsymbol{S}^{(1)} = \sigma\left(\frac{1}{\sqrt{d_k}}\boldsymbol{Q}\boldsymbol{K}^T\right)$ and $\boldsymbol{W}^{(1)} = \boldsymbol{W}^{(v)}$ for simplicity of notation. Figure 11.5 illustrates (11.9) in the TN notation.

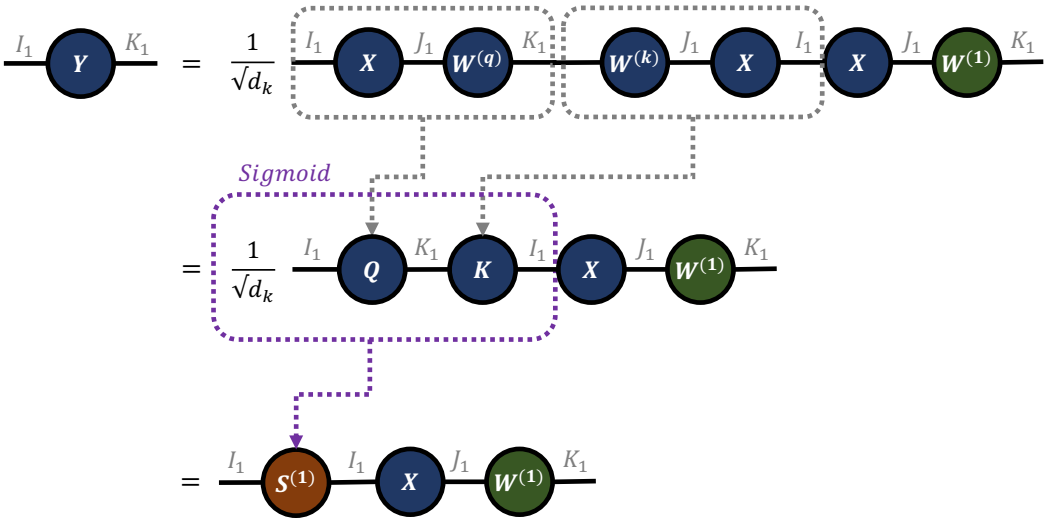


Figure 11.5: ANN as a GTN with $N = 1$ domain mode (representing time) and $M = 1$ feature mode, according to (11.9).

Remark 37 *The GSO, $\boldsymbol{S}^{(1)} = \sigma\left(\frac{1}{\sqrt{d_k}}\boldsymbol{Q}\boldsymbol{K}^T\right)$, where $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^{(q)}$ and $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{(k)}$, can be interpreted as the adjacency matrix of a time-domain graph, where the vertices represent different time-steps. In this graph, the edge weight connecting pairs of time-steps is inferred through a pair-wise similarity function parameterized by trainable weight matrices $\boldsymbol{W}^{(q)}$ and $\boldsymbol{W}^{(k)}$. This allows the attention mechanism to construct a different graph topology depending on the input itself, which is the core of the self-attention mechanism.*

11.6.5 *Recurrent Neural Network*

A RNN [69] takes as input an order-$(1+1)$ data tensor (a time-series features matrix with $N = 1$ time-domain mode and $M = 1$ feature mode), $\boldsymbol{X} \in \mathbb{R}^{I_1 \times J_1}$, where the i_1-th row vector, $\boldsymbol{x}_{i_1} \in \mathbb{R}^{J_1}$, is the feature vector at the i_1-th time-step. The RNN is characterized by the recurrent computation of hidden states

$$\boldsymbol{y}_{i_1} = \boldsymbol{W}^{(r)}\boldsymbol{y}_{i_1-1} + \boldsymbol{W}^{(x)}\boldsymbol{x}_{i_1} \tag{11.10}$$

where $\boldsymbol{W}^{(r)} \in \mathbb{R}^{K_1 \times K_1}$ is a trainable recurrent weight matrix, $\boldsymbol{y}_{i_1-1} \in \mathbb{R}^{K_1}$ is the hidden state from the previous time-step, $\boldsymbol{W}^{(x)} \in \mathbb{R}^{K_1 \times J_1}$ is a trainable input weight matrix, and $\boldsymbol{x}_{i_1} \in \mathbb{R}^{J_1}$ is the input feature vector at the current time-step. The hidden state equation in (11.10) can be shown to be a special case of (11.1) with $N = 1$ domain mode (representing time) and $M = 1$ feature mode. This is achieved by

expanding the recurrent computation as

$$\begin{aligned}\boldsymbol{y}_{i_1} &= \boldsymbol{W}^{(r)}\boldsymbol{y}_{i_1-1} + \boldsymbol{W}^{(x)}\boldsymbol{x}_{i_1} \\ &= \boldsymbol{W}^{(r)}\boldsymbol{y}_{i_1-1} + \tilde{\boldsymbol{x}}_{i_1} \\ &= \boldsymbol{W}^{(r)}\left(\boldsymbol{W}^{(r)^{i_1-2}}\tilde{\boldsymbol{x}}_1 + \cdots + \tilde{\boldsymbol{x}}_{i_1-1}\right) + \tilde{\boldsymbol{x}}_{i_1} \\ &= \boldsymbol{W}^{(r)^{i_1-1}}\tilde{\boldsymbol{x}}_1 + \boldsymbol{W}^{(r)^{i_1-2}}\tilde{\boldsymbol{x}}_2 + \cdots + \tilde{\boldsymbol{x}}_{i_1}\end{aligned} \tag{11.11}$$

where $\tilde{\boldsymbol{x}}_{i_1} = \boldsymbol{W}^{(x)}\boldsymbol{x}_{i_1}$ for simplicity of notation. This can then be written in a block-matrix form for I_1 time-steps as

$$\begin{bmatrix}\boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_{I_1}\end{bmatrix} = \begin{bmatrix}\boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{W}^{(r)^1} & \boldsymbol{I} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{W}^{(r)^{I_1-1}} & \boldsymbol{W}^{(r)^{I_1-2}} & \cdots & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\tilde{\boldsymbol{x}}_1 \\ \tilde{\boldsymbol{x}}_2 \\ \vdots \\ \tilde{\boldsymbol{x}}_{I_1}\end{bmatrix} \tag{11.12}$$

By allowing the recurrent weight matrix to be a scaled idempotent matrix, $\boldsymbol{W}^{(r)} = c\boldsymbol{W}^{(1)}$, such that $\boldsymbol{W}^{(r)^N} = c^N\boldsymbol{W}^{(1)}$, the expression (11.12) can be simplified into

$$\begin{bmatrix}\boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_{I_1}\end{bmatrix} = \begin{bmatrix}\boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ c^1\boldsymbol{W} & \boldsymbol{I} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ c^{I_1-1}\boldsymbol{W} & c^{I_1-2}\boldsymbol{W} & \cdots & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\tilde{\boldsymbol{x}}_1 \\ \tilde{\boldsymbol{x}}_2 \\ \vdots \\ \tilde{\boldsymbol{x}}_{I_1}\end{bmatrix} \tag{11.13}$$

Observe that expression (11.13) can be written compactly through the Kronecker product as $\operatorname{vec}(\boldsymbol{Y}) = \left(\boldsymbol{S}^{(1)} \otimes \boldsymbol{W}^{(1)} + \boldsymbol{I}\right)\operatorname{vec}\left(\tilde{\boldsymbol{X}}\right)$, where $\boldsymbol{S}^{(1)}_{i_1,i_2} = c^{i_1-i_2}$ for $i_1 > i_2$ and $\boldsymbol{S}^{(1)}_{i_1,i_2} = 0$ otherwise. By using the vectorized Tucker product property in Section 2.1, the matricization of (11.13) can be written as

$$\begin{aligned}\boldsymbol{Y} &= \operatorname{mat}\left(\left(\boldsymbol{S}^{(1)} \otimes \boldsymbol{W}^{(1)} + \boldsymbol{I}\right)\operatorname{vec}\left(\tilde{\boldsymbol{X}}\right)\right) \\ &= \operatorname{mat}\left(\left(\boldsymbol{S}^{(1)} \otimes \boldsymbol{W}^{(1)}\right)\operatorname{vec}\left(\tilde{\boldsymbol{X}}\right) + \operatorname{vec}\left(\tilde{\boldsymbol{X}}\right)\right) \\ &= \operatorname{mat}\left(\operatorname{vec}\left(\left[\!\left[\tilde{\boldsymbol{X}}; \boldsymbol{S}^{(1)}, \boldsymbol{W}^{(1)}\right]\!\right]\right) + \operatorname{vec}\left(\tilde{\boldsymbol{X}}\right)\right) \\ &= \operatorname{mat}\left(\operatorname{vec}\left(\left[\!\left[\tilde{\boldsymbol{X}}; \boldsymbol{S}^{(1)}, \boldsymbol{W}^{(1)}\right]\!\right] + \tilde{\boldsymbol{X}}\right)\right) \\ &= \left[\!\left[\tilde{\boldsymbol{X}}; \boldsymbol{S}^{(1)}, \boldsymbol{W}^{(1)}\right]\!\right] + \tilde{\boldsymbol{X}}\end{aligned} \tag{11.14}$$

This corresponds to the forward pass in (11.1) with a pre-processed input, $\tilde{\boldsymbol{X}} = \boldsymbol{W}^{(x)}\boldsymbol{X}^T$, and a skip-connection. Figure 11.6 illustrates (11.14) in the TN notation.

Remark 38 *The GSO, $\boldsymbol{S}^{(1)} \in \mathbb{R}^{I_1 \times I_1}$, of an RNN can be interpreted as the graph adjacency matrix of a time-domain graph, where the I_1 vertices represent consecutive time-steps, while an edge connecting two time-steps has an edge weight that decays as a function of their time-gap, $\boldsymbol{S}^{(1)}_{i_1,i_2} = c^{i_1-i_2}$ for $i_1 > i_2$. Therefore, the further away two time-steps are, the weaker the connection between them. The condition $\boldsymbol{S}_{i_1,i_2} = 0$ for $i_1 \leq i_2$ ensures the directed flow of time, where the past time-steps can influence futures states, but not vice-versa.*

Remark 39 *If we constrain $\boldsymbol{W}^{(1)} = \boldsymbol{I}$, then the recurrent computation of hidden states in (11.10) simplifies to $\boldsymbol{y}_{i_1} = c\boldsymbol{y}_{i_1-1} + \boldsymbol{W}^{(x)}\boldsymbol{x}_{i_1}$, which in turn reduces (11.14) to the un-normalized GCN forward pass, given by*

$$\begin{aligned} \boldsymbol{Y} &= [\![\tilde{\boldsymbol{X}}; \boldsymbol{S}^{(1)}, \boldsymbol{I}]\!] + \tilde{\boldsymbol{X}} \\ &= \boldsymbol{S}^{(1)}\tilde{\boldsymbol{X}} + \tilde{\boldsymbol{X}} \\ &= \left(\boldsymbol{I} + \boldsymbol{S}^{(1)}\right)\tilde{\boldsymbol{X}} \end{aligned} \tag{11.15}$$

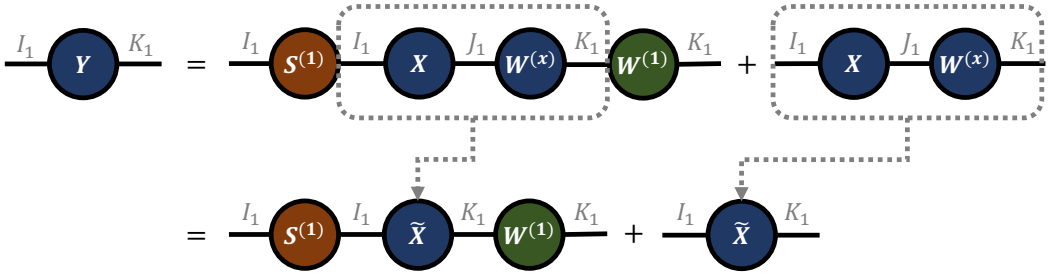


Figure 11.6: RNN as a GTN with $N = 1$ domain mode (representing time) and $M = 1$ feature mode, according to (11.14).

11.7 Beyond Classical Neural Networks

As discussed in Section 11.6, classical neural network architectures are typically designed to process data with only one domain mode. However, modern data sources often reside on multiple irregular domains, such as video data (pixel data over time), traffic data (time-series data over graph), and social network data (natural language data over graph). Designing a neural network architectures for processing data over multiple domains is particularly challenging, both in terms of handling multiple data dimensions jointly and digesting the sheer data volume. Using classical neural network architectures, this will likely require a combination of RNNs, CNNs, and GNNs that will inevitably explode in size due to the high-dimensionality of the problem. However, the considered data can be readily handled under the proposed GTN framework, which bypasses the cumbersome handling of multiple dimensions through intuitive TN diagrams. Indeed, a data tensor residing on N domains simultaneously admits a simple representation as a Tucker product with N GSOs according to (11.1), as illustrated in Figure 11.1.

11.8 Experiments

To illustrate the generality and flexibility of the proposed GTN framework for designing neural learning systems on multiple domains, a traffic modelling experiment was considered. Traffic modelling requires the processing of large-scale traffic data on both the time domain and the sensor network domain, which makes it a natural candidate for the proposed framework.

Remark 40 *The goal of the experiment is to illustrate how neural network architectures for processing data on multiple domains can be readily designed using the proposed GTN framework. In practice, the performance of the GTN model will vary significantly depending on the design of the GSO. The design of optimal GSO for traffic modelling is beyond the scope of this work and is left to domain experts.*

Dataset: The PEMS-BAY traffic dataset [166] was considered for the experiment, which is a traffic speed dataset collected from the California Transportation Agency. This dataset contains the average traffic speed measured across 325 sensors in the Bay area from 2017-Jan-01 to 2017-May-31 sampled at a 5 minutes intervals, resulting in a total of $52,116$ time-steps.

Experiment Settings: We closely followed the experiment procedure detailed in [167]. More specifically, the dataset was split chronologically such that the first 70% was used for training, while the remaining 10% and 20% were respectively used for validation and testing. The samples were then generated using a rolling window of 24 time-steps (equivalent to 2 hours), such that the first 12 time-steps were used as input (equivalent to 1 hour) while the last 12 time-steps were used as output labels (i.e., 1 hour prediction horizon). The data was scaled to zero-mean and unit-variance. The tuned model that achieved the best performance within the validation set was then applied to the test set for out-of-sample evaluation.

Data Domain GSOs: The PEMS-BAY traffic dataset contains vehicle speed measurements across (i) the time-domain and (ii) the sensors network domain. To process the speed data over time, a time-domain GSO was designed according to the procedure outlined in Section 11.6.5. On the other hand, to process speed data over sensors, a sensors network GSO was constructed following the procedure detailed in [167]. More specifically, a graph adjacency matrix was constructed using a pair-wise distance between sensor locations via a threshold Gaussian kernel, $\boldsymbol{S}_{i,j} = exp\left(\frac{-\text{dist}(v_i,v_j)^2}{\sigma^2}\right)$ if $\text{dist}(v_i, v_j) \leq k$ and $\boldsymbol{S}_{i,j} = 0$ otherwise, where $\boldsymbol{S}_{i,j}$ denotes the (i,j)-th entry of the GSO associated with the i-th and j-th sensors. In this way, the sensor network GSO reflects the physical proximity information between all pairs of sensors.

GTN Model: The considered input samples were effectively order-$(2+0)$ data tensors, $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$, with $N = 2$ domain modes and $M = 0$ feature modes. The time-domain mode consists of $I_1 = 12$ time-steps while the sensors-domain mode consists of $I_2 = 325$ sensors. By initializing the data domain GSOs as discussed in the previous paragraph, a corresponding GTN layer can be straightforwardly designed according to (11.1). For this experiment, we constructed a model by stacking 4 such GTN layers with the *tanh* activation, followed by an output layer. The output layer consists of: (i) a weight vector, $\boldsymbol{w} \in \mathbb{R}^{I_1}$, which computes a weighted sum over the time-mode, and (ii) a sensor-domain GSO applied once again to the sensors mode. Figure 11.7 illustrates the design of the considered model in the TN notation. The considered model was implemented in TensorFlow and trained using a Mean-Absolute-Error

(MAE) loss function with an Adam optimizer, with a learning rate of 0.001 over 100 epochs.

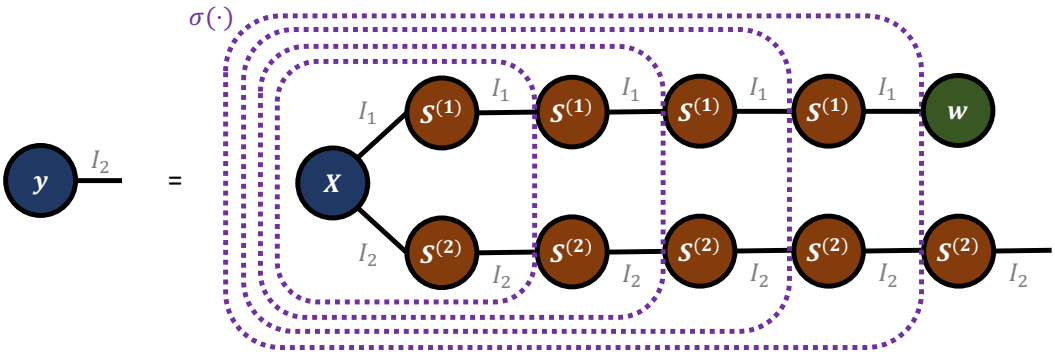


Figure 11.7: GTN model for traffic modelling. The input data tensor, $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$, contains speed measurements across $I_1 = 12$ time-steps and $I_2 = 325$ sensors. This is then contracted according to the GTN forward pass in (11.1) with a time-domain GSO, $\boldsymbol{S}^{(1)} \in \mathbb{R}^{I_1 \times I_1}$, and a sensors-domain GSO, $\boldsymbol{S}^{(2)} \in \mathbb{R}^{I_2 \times I_2}$, followed by a $tanh$ activation, $\sigma(\cdot)$. The considered GTN forward pass is then repeated 4 times. Finally, the output layer applies a time-mode contraction with a weight vector, $\boldsymbol{w} \in \mathbb{R}^{I_1}$, and a sensors-mode contraction with the sensors-mode GSO, $\boldsymbol{S}^{(2)} \in \mathbb{R}^{I_2 \times I_2}$, to yield the final output vector, $\boldsymbol{y} \in \mathbb{R}^{I_2}$. The resulting output contains the forecast of the traffic speeds for the next hour across $I_2 = 325$ sensors.

Model Evaluation: The proposed model was evaluated against the same set of models as in [167], which includes: Historical Average (HA); Auto-Regressive Integrated Moving-Average (ARIMA); Vector Auto-Regression (VAR); Support Vector Regression (SVR); DNN; Long Short-Term Memory network with Fully-Connected hidden units (FC-LSTM) [168]; Diffusion Convolutional RNN (DCRNN) [167]. In addition, for rigour, we also evaluated the proposed model against tensor learning algorithms such as the CPD TR model [23].

	GTN	HA	ARIMA	VAR	SVR	DNN	FC-LSTM	DCRNN	TR
MAE	2.41	2.88	3.38	2.93	3.28	2.46	2.37	2.07	2.76
RMSE	5.12	5.59	6.50	5.44	7.08	4.98	4.96	4.74	5.48
MAPE	5.7%	6.8%	8.3%	6.5%	8.0%	5.9%	5.7%	4.9%	6.5%

Table 11.1: Traffic prediction performance achieved by considered models.

Results: The experiment results are summarised in Table 11.1. The proposed GTN model was able to achieve superior performance compared to most general purpose benchmark models such as HA, ARIMA, VAR, SVR, and DNN. Only models that were

specifically designed to process spatial-temporal data such as FC-LSTM and DCRNN were able to outperform the the proposed model.

11.9 Summary

We have introduced a graphical, tensor-based framework for describing and designing neural networks through a joint consideration of graphs and TNs. The so introduced GTN framework has been shown to be general enough to include many modern neural network architectures as special cases, while being flexible enough to allow for the processing of data over any and many domains. The power and flexibility of the proposed framework has been demonstrated through several real-data case studies. It is our hope that by offering a common graphical language to formally describe neural networks, this will open avenues for including sophisticated tensor techniques developed in other fields, such as quantum many-body physics, for designing new classes of neural networks with superior performance and complexity characteristics.

Chapter 12

Conclusions and Future Work

This thesis illustrates the feasibility and potential of a conjoint treatment of graphs and tensors in several data analytics applications. After demonstrating the merits of tensors and graphs as individual tools for solving ML problems in **Part I** and **Part II** respectively, several novel ML frameworks have been introduced in **Part III**, which simultaneously leverages on the ability of graphs to handle irregular data, and the super-compression properties of tensors to reduce complexity costs.

In particular, the following approaches have been introduced: (i) GRTR framework for modelling high-dimensional tensor data while accounting for the underlying graph structure; (ii) Tensor-algebraic approach for computing efficient convolution on graphs; (iii) GTN framework for designing neural learning systems which is both general enough to describe most existing neural network architectures and flexible enough to model large-dimensional data on any and many irregular domains. Throughout this thesis, the proposed frameworks have been shown to exhibit desirable properties in terms of expressive power, interpretability, and complexity, across a range of application domains and learning paradigms, demonstrating its versatility and potential in many real-life experimental settings.

While this thesis primarily focuses on the enhancement of ML algorithms through the integration of tensors and graphs, there is scope for future research to explore the amalgamation of these techniques with the tensorized representation of high-dimensional data that possesses a relational structure. By pursuing this line of inquiry, researchers can potentially develop next generation of data analytics tools that further accelerates on the considered methods, as they leverage tensorization throughout the entire data processing chain.

To conclude, an overview of the research achievements in this thesis as well as potential future research directions is presented in Section 12.1.

12.1 Research Achievements and Future Work

The research achievements and potential future research directions are summarised and presented below for each chapter.

Part I: Tensor Learning Methods

- **Chapter 3: Tensor Contraction via Tensor-Train Decomposition**

  Research Achievements. A key tool underpinning multi-linear manipulation of tensors and TNs is the standard TCP. However, depending on the dimensionality of the underlying tensors, the TCP also comes at the price of an exponential increase in computational complexity in tensor order. To address this issue, we employ diagrammatic TN manipulation in order to calculate such products in an efficient, maths-free, and computationally tractable manner. This has been achieved by making use of TTD, which has rendered the underlying concepts straightforward to interpret and implement, thereby enhancing intuition of the associated multilinear operations, while preserving mathematical rigour. In addition to completely bypassing the cumbersome mathematical multi-linear expressions and thus eliminating the main source of errors in tensor manipulation, the introduced TTCP model has been shown to accelerate significantly the underlying computational operations, as it is independent of the tensor order for tensors stored in the distributed TT format.

  Future Work. While this work focuses on the single-mode contraction between two tensors in the TT format, practical applications often require multiple contractions across multiple tensors simultaneously. Future research could therefore extend the considered framework to such scenarios.

- **Chapter 4: Tensor-Train Recurrent Neural Networks for Finance**

  Research Achievements. Traditional currency forecasting models are typically based on intra-class variables only, despite the wide availability of other strongly or weakly related asset classes. To address this issue, we investigated the ability of TTRNNs to perform financial forecasting of currencies from multi-class data. This has been achieved based on multi-way financial data from four asset classes: equities, currencies, commodities, and fixed income. The model has been trained on tensorized data from May 2006 to February 2018, and tested on the period from February 2018 to May 2019. Simulation results have verified the desirable compression and regularization properties of the TTRNN, which outperformed its uncompressed RNN counterpart and other tensor based forecasting methods. Moreover, through the gradient inspection of individual TT cores, TTRNN has been demonstrated to allow for physical interpretability of relative modal importance in a financial context, thus mitigating the well-known black-box issue inherent to neural networks.

  Future Work. This work focuses on the application of TTD for reducing the complexity of RNNs in the context of financial modelling. However, depending on the type of TD, the performance of such models will vary greatly. Therefore, future research could explore the complexity and performance trade-offs of RNNs compressed via different TD techniques. In addition, the proposed mode-wise interpretability method can be further explored in conjunction with other time-series feature importance methods such as Shapley values [169] for further insights.

Part II: Graph Learning Methods

- **Chapter 5: Graph Theory for Metro Traffic Modelling**

  Research Achievements. We have introduced a graph-theoretic framework which makes it possible to assess the functional design and operation of public transport networks in a mathematically tractable and intuitive manner. The formulation allows not only for an assessment of the functionality, but also offers an opportunity to facilitate enhancements and assess vulnerabilities, notwithstanding the possibility of a dynamic management of the network. Our focus has been on establishing a link between the laws of diffusive fields and the parameters of urban underground traffic. In this way, we have been able to leverage on the ability of graphs to represent data on irregular domains to introduce an intuitive model for the population migration during the rush hours. Furthermore, we have employed the hyper-graph representation of the network connections, and have demonstrated that this accounts for multiple links between underground stations in a natural way, which is not achievable based on the standard graph model. Finally, we have explored the connection between graphs and modern GNNs models in this context, demonstrating the power of this approach on the prediction of the evening traffic based on the morning traffic data. It is our hope to have provided a platform to approach both traffic planning and urban transition issues from a rigorous engineering perspective using GSP techniques.

  Future Work. This work applies graph-theoretic tools in the context of metro traffic modelling. Given the nature of the dataset, temporal information about passenger flow can also be included into the model to improve the performance. This requires the use of spatial-temporal models that can jointly process information over the time-domain and the network-domain. Future research can explore the application of such models for modelling the London undergound network.

- **Chapter 6: Hierarchical Graph Pooling for Finance**

  Research Achievements. We have introduced a finance-specific graph pooling operator, referred as the *StockPool*, with the aim to improve computational efficiency while maintaining enhanced end-to-end performance, compared to existing pooling operations. This has been achieved by incorporating domain knowledge related to the stock graph into the graph coarsening process. Such a domain-aware graph pooling operator has been developed by clustering stocks into their corresponding industries, under the assumption that such stocks share key underlying economic features. We have also investigated the use of different graph pooling operators as a basis for constructing highly uncorrelated GNN models, a pre-requisite for a successful ensemble model. Experimentation based on the S&P500 stock index have demonstrated that the graph ensemble model outperforms its constituent sub-models, as well as the general movement of the market.

  Future Work. The proposed *StockPool* leverages on the underlying economics of the considered stocks to perform deterministic pooling. This concept can be

easily extended to handle asset classes other than equity, such as currencies, commodities, and fix-income securities. Future work could investigate such applications, which would allow for the construction of cross-asset investment strategies.

Part III: Tensor-Variate Machine Learning on Graphs

- **Chapter 7: Graph-Regularized Tensor Regression**

  Research Achievements. We have introduced a TR framework with graph-based regularization, which makes it possible to efficiently incorporate domain knowledge into large-scale, multi-way regression through graph Laplacian regularization. The GRTR model has been shown to be both physically meaningful and to provide enhanced results against standard vector and TR models, both in terms of out-of-sample performance and computational costs. By virtue of tensor algebra, the proposed model has also been shown to enable enhanced interpretability and control over the learning process. The proposed GRTR framework is general and opens avenues for exploring numerous possibilities for formulating physically meaningful models that accounts for intral-modal relations in big data applications.

  Future Work. The performance of the proposed GRTR is highly dependent on the design of the associated GSOs. Therefore, future research could investigate data-driven methods for designing GSOs. This leverage on mode-wise correlations or the application of deep learning methods such as attention.

- **Chapter 8: Graph Convolution via Convolution Tensors**

  Research Achievements. We have introduced a graph convolution method based on the convolution tensor, which offers several advantages over existing methods in terms of efficiency, scalability, robustness, flexibility and performance. The so proposed GCTN is also shown to be general enough to include the classical CNN as a special case of convolution over circulant graphs. The benefits of the GCTN are validated through experiments, achieving state-of-the-art performance on the protein classification dataset.

  Future Work. For a given graph domain, the proposed convolution tensor is constructed through a deterministic algorithm given the associated graph adjacency matrix. However, the convolution tensor can also be learned directly from the data without requiring the knowledge of the underlying graph topology. This would allow for the application of the considered framework to datasets where the underlying structure is not obvious. Future research could therefore explore the use of data-driven methods (such as Reinforcement Learning [170]) for learning the convolution tensor directly.

- **Chapter 9: Recurrent Graph Tensor Networks**

  Research Achievements. We have introduced a novel RGTN framework for modelling time-series data, by combining the expressive power of TNs with

the ability of graphs to account for the structure underlying time-series data. Experiment results have verified the desirable properties of the proposed RGTN framework, which outperformed RNN, GRU, LSTM, and TTRNN models across multiple time-series modelling tasks, and at a drastically reduced parameter complexity.

Future Work. The proposed framework applies tensor- and graph-theoretic concepts to describe the inner workings of RNNs. Future research could extend the considered framework to other RNN variants, such as LSTMs and GRUs.

- **Chapter 10: Attention Graph Tensor Networks**

  Research Achievements. We have introduced a novel TSA mechanism for NLP. By virtue of its conjoint tensor and graph formulation, the proposed framework has been shown to achieve high expressive power at a drastically lower parameter complexity compared to traditional attention models. Experimental results have verified the desirable properties of the proposed TSA, with better or comparable performance achieved across five NLP benchmark tasks at varying levels of complexity. The performance gain has been demonstrated to be especially fruitful in low complexity cases.

  Future Work. The proposed framework considers input tokens embedded in the tensor domain to improve the parameter complexity of the weight matrices by formulating them as MPOs. Future research could explore the storage of input token tensors in MPS form directly. This would allow for the end-to-end formulation of the attention mechanism in TT form, which could further improve the time and space complexity of transformers.

- **Chapter 11: Generalized Graph Tensor Networks**

  Research Achievements. We have introduced a graphical, tensor-based framework for describing and designing neural networks through a joint consideration of graphs and TNs. The so introduced GTN framework has been shown to be general enough to include many modern neural network architectures as special cases, while being flexible enough to allow for the processing of data over any and many domains. The power and flexibility of the proposed framework has been demonstrated through several real-data case studies. It is our hope that by offering a common graphical language to formally describe neural networks, this will open avenues for including sophisticated tensor techniques developed in other fields, such as quantum many-body physics, for designing new classes of neural networks with superior performance and complexity characteristics.

  Future Work. The proposed framework is general enough to incorporate many existing neural network architectures as special cases, where the primary difference is captured by the design of the underlying GSOs. Future research could explore novel algorithms for neural architecture search based on the proposed GTN framework. This could help bypass the current trial-and-error approach to neural network design.

Bibliography

[1] Y. L. Xu, K. Konstantinidis, and D. P. Mandic, "Graph-regularized tensor regression: A domain-aware framework for interpretable modelling of multi-way data on graphs." *Neural Computation, in print*, 2023. pages 20, 33

[2] I. Kisil, G. G. Calvi, K. Konstantinidis, Y. L. Xu, and D. P. Mandic, "Accelerating tensor contraction products via tensor-train decomposition," *IEEE Signal Processing Magazine*, vol. 39, no. 5, pp. 63–70, 2022. pages 20, 30, 148

[3] T. Konstantinidis, Y. L. Xu, T. G. Constantinides, and D. P. Mandic, "A comparative study on ML-based approaches for main entity detection in financial reports," in *Proceedings of the 24th International Conference on Digital Signal Processing (DSP)*, 2023. pages 20

[4] T. Konstantinidis, M. Golan, Y. L. Xu, T. G. Constantinides, and D. P. Mandic, "Text mining for sentiment analysis in bond portfolio construction," in *Proceedings of the 24th International Conference on Digital Signal Processing (DSP)*, 2023. pages 20

[5] T. Konstantinidis, A. Bonet-Farres, Y. L. Xu, T. G. Constantinides, and D. P. Mandic, "Financial news classification model for NLP-based bond portfolio construction," in *Proceedings of the 24th International Conference on Digital Signal Processing (DSP)*, 2023. pages 20

[6] A. N. Arya, Y. L. Xu, L. Stankovic, and D. P. Mandic, "Hierarchical graph learning for stock market prediction via a domain-aware graph pooling operator," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023. pages 21, 32

[7] A. Rebello, K. Konstantinidis, Y. L. Xu, and D. P. Mandic, "Tensor completion for efficient and accurate hyperparameter optimisation in large-scale statistical learning," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023. pages 21

[8] J. Roa-Vicens, Y. L. Xu, R. Silva, and D. P. Mandic, "Graph and tensor-train recurrent neural networks for high-dimensional models of limit order books," in *Proceedings of Third ACM International Conference on AI in Finance (ICAIF)*, 2022, pp. 207–213. pages 21, 34

[9] K. Konstantinidis, Y. L. Xu, Q. Zhao, and D. P. Mandic, "Variational Bayesian tensor networks with structured posteriors," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 3638–3642. pages 21

[10] Y. L. Xu, K. Konstantinidis, S. Li, L. Stankovic, and D. P. Mandic, "Low-complexity attention modelling via graph tensor networks," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 3928–3932. pages 21, 34, 93, 149

[11] Y. L. Xu and D. P. Mandic, "Recurrent graph tensor networks: A low-complexity framework for modelling high-dimensional multi-way sequences," in *Proceedings of the 29th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 1795–1799. pages 21, 34, 93, 136, 138, 139, 148

[12] B. Scalzo, Y. L. Xu, A. G. Constantinides, and D. P. Mandic, "Graph theory for metro traffic modelling," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–5. pages 21, 25, 32, 83

[13] Y. L. Xu, G. G. Calvi, and D. P. Mandic, "Tensor-train recurrent neural networks for interpretable multi-way financial forecasting," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–5. pages 21, 31, 91, 149

[14] Y. L. Xu, K. Konstantinidis, and D. P. Mandic, "A tensorized spectral attention mechanism for efficient natural language processing," in *the Second Workshop on Quantum Tensor Networks in Machine Learning, 35th Conference on Neural Information Processing Systems (NIPS)*, 2021. pages 21, 34

[15] K. Konstantinidis, Y. L. Xu, and D. P. Mandic, "Bayesian tensor networks with structured posteriors," in *the Second Workshop on Quantum Tensor Networks in Machine Learning, 35th Conference on Neural Information Processing Systems (NIPS)*, 2021. pages 22

[16] Y. L. Xu, K. Konstantinidis, and D. P. Mandic, "Multi-graph tensor networks," in *the First Workshop on Quantum Tensor Networks in Machine Learning, 34th Conference on Neural Information Processing Systems (NIPS)*, 2020. pages 22, 28, 35, 91, 92, 93, 136, 139, 148

[17] A. Cichocki, "Era of big data processing: A new approach via tensor networks and tensor decompositions," in *Proceedings of the International Workshop on Smart Info-Media Systems in Asia*, Mar. 2014. pages 23, 39, 42, 91, 123

[18] A. Cichocki, N. Lee, I. Oseledets, A. Phan, Q. Zhao, D. P. Mandic *et al.*, "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions," *Foundations and Trends in Machine Learning*, vol. 9, no. 4-5, pp. 249–429, 2016. pages 23, 37, 42, 46, 51, 52, 62, 92, 126, 147

[19] A. Cichocki, A. Phan, Q. Zhao, N. Lee, I. Oseledets, M. Sugiyama, D. P. Mandic *et al.*, "Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives," *Foundations and Trends in Machine Learning*, vol. 9, no. 6, pp. 431–673, 2017. pages 23, 37, 62, 92, 111, 126, 147

[20] A. Cichocki, D. Mandic, L. D. Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, ""Tensor decompositions for signal processing applications: From two-way to multiway component analysis"," *IEEE Signal Processing Magazine*, vol. 32, pp. 145–163, 2015. pages 23, 24, 61, 92, 147

[21] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009. pages 23, 61

[22] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017. pages 23, 61

[23] W. Guo, I. Kotsia, and I. Patras, "Tensor learning for regression," *IEEE Transactions on Image Processing*, vol. 21, no. 2, pp. 816–827, 2012. pages 23, 92, 93, 94, 149, 159

[24] H. Zhou, L. Li, and H. Zhu, "Tensor regression with applications in neuroimaging data analysis," *Journal of the American Statistical Association*, vol. 108, no. 502, pp. 540–552, 2013. pages 23, 92, 93

[25] X. Li, D. Xu, H. Zhou, and L. Li, "Tucker tensor regression and neuroimaging analysis," *Statistics in Biosciences*, vol. 10, no. 3, pp. 520–545, 2018. pages 23, 92, 93

[26] M. Hou and B. Chaib-Draa, "Hierarchical Tucker tensor regression: Application to brain imaging data analysis," in *Proceedings of 2015 IEEE International Conference on Image Processing (ICIP)*, 2015, pp. 1344–1348. pages 23, 92, 93

[27] Y. Liu, J. Liu, and C. Zhu, "Low-rank tensor train coefficient array estimation for tensor-on-tensor regression," *IEEE Transactions on Neural Networks and Learning Systems*, 2020. pages 24, 92, 93

[28] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 442–450. pages 24, 27, 62, 63, 66, 93, 122, 123, 127, 128, 149

[29] Y. Yang, D. Krompass, and V. Tresp, "Tensor-train recurrent neural networks for video classification," *in Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 3891–3900, 2017. pages 24, 62, 63, 123, 127, 149

[30] N. Cohen, O. Sharir, and A. Shashua, "On the expressive power of deep learning: A tensor analysis," in *Proceedings of the Conference on Learning Theory*, 2016, pp. 698–728. pages 24, 51, 122, 149

[31] A. Padia, K. Kalpakis, and T. Finin, "Inferring relations in knowledge graphs with tensor decompositions," in *Proceedings of the IEEE International Conference on Big Data*, 2016, pp. 4020–4022. pages 24, 51

[32] M. Nickel, V. Tresp, and H. P. Kriegel, "Factorizing YAGO: Scalable machine learning for linked data," in *Proceedings of the 21st International Conference on World Wide Web*, 2012, pp. 271–280. pages 24, 51

[33] M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017. pages 24, 25, 49, 109, 122, 140

[34] L. Stankovic, D. Mandic, M. Dakovic, M. Brajovic, B. Scalzo, and A. G. Constantinides, "Data analytics on graphs. Part II: Signals on graphs," *Foundations and Trends in Machine Learning*, vol. 13, no. 2–3, pp. 158–331, 2020. pages 25, 37, 48, 49, 71, 76, 92, 122

[35] L. Stankovic, D. Mandic, M. Dakovic, M. Brajovic, B. Scalzo, and T. Constantinides, "Data analytics on graphs. Part I: Graphs and spectra on graphs," *Foundations and Trends in Machine Learning*, vol. 13, no. 1, pp. 1–157, 2020. pages 25, 37, 71, 72, 76, 92, 122

[36] L. Stankovic, D. Mandic, M. Dakovic, M. Brajovic, B. Scalzo, S. Li, and A. G. Constantinides, "Data analytics on graphs. Part III: Machine learning on graphs, from graph topology to applications," *Foundations and Trends in Machine Learning*, vol. 13, no. 4, pp. 332–530, 2020. pages 25, 28, 37, 71, 81, 84, 92, 122

[37] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020. pages 25, 110, 122, 123, 141

[38] T. Bai, Y. Zhang, B. Wu, and J.-Y. Nie, "Temporal graph neural networks for social recommendation," in *Proceedings of the IEEE International Conference on Big Data (Big Data)*, 2020, pp. 898–903. pages 25, 83

[39] Z. Lu, W. Lv, Z. Xie, B. Du, and R. Huang, "Leveraging graph neural network with LSTM for traffic speed prediction," in *Proceedings of the IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*, 2019, pp. 74–81. pages 25, 83

[40] Y. Yuan and Z. Bar-Joseph, "GCNG: Graph convolutional networks for inferring gene interaction from spatial transcriptomics data," *Genome biology*, vol. 21, no. 1, pp. 1–16, 2020. pages 25, 83

[41] J. A. Schouten, *Ricci-calculus: an introduction to tensor analysis and its geometrical applications*. Springer Science & Business Media, 2013, vol. 10. pages 27

[42] J. Stachel, D. K. Buchwald, and P. G. Bergmann, *The collected papers of Albert Einstein*. Princeton University Press Princeton, 1987, vol. 1. pages 27

[43] R. A. Harshman, "Foundations of the parafac procedure: Models and conditions for an explanatory multimodal factor analysis," 1970. pages 27

[44] F. L. Hitchcock, "Multiple invariants and generalized rank of a p-way matrix or tensor," *Journal of Mathematics and Physics*, vol. 7, no. 1-4, pp. 39–79, 1928. pages 27, 44, 93

[45] J. Douglas Carroll, S. Pruzansky, and J. B. Kruskal, "Candelinc: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters," *Psychometrika*, vol. 45, pp. 3–24, 1980. pages 27

[46] C. J. Appellof and E. R. Davidson, "Strategies for analyzing data from video fluorometric monitoring of liquid chromatographic effluents," *Analytical Chemistry*, vol. 53, no. 13, pp. 2053–2056, 1981. pages 27

[47] L. R. Tucker, "Implications of factor analysis of three-way matrices for measurement of change," *Problems in measuring change*, vol. 15, no. 122-137, p. 3, 1963. pages 27

[48] W. Hackbusch and S. Kühn, "A new scheme for the tensor representation," *Journal of Fourier analysis and applications*, vol. 15, no. 5, pp. 706–722, 2009. pages 27

[49] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011. pages 27, 42, 43, 54, 62, 123, 137

[50] G. Vidal, "Entanglement renormalization," *Physical review letters*, vol. 99, no. 22, p. 220405, 2007. pages 27

[51] J. F. Pedraza, A. Russo, A. Svesko, and Z. Weller-Davies, "Sewing spacetime with lorentzian threads: complexity and the emergence of time in quantum gravity," *Journal of High Energy Physics*, vol. 2022, no. 2, pp. 1–123, 2022. pages 27

[52] F. Jamet, C. Lenihan, L. P. Lindoy, A. Agarwal, E. Fontana, B. A. Martin, and I. Rungger, "Anderson impurity solver integrating tensor network methods with quantum computing," *arXiv preprint arXiv:2304.06587*, 2023. pages 27

[53] L. Euler, "Solutio problematis ad geometriam situs pertinentis," *Commentarii academiae scientiarum Petropolitanae*, pp. 128–140, 1741. pages 27

[54] D. B. West, *Introduction to graph theory*. Prentice hall Upper Saddle River, 2001, vol. 2. pages 27

[55] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011. pages 27

[56] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013. pages 27

[57] D. M. Cvetkovic, "Spectra of graphs. theory and application," 1980. pages 27

[58] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017. pages 27, 110, 152

[59] K. G. Derpanis and R. P. Wildes, "Classification of traffic video based on a spatiotemporal orientation analysis," in *Proceedings of the 2011 IEEE Workshop on Applications of Computer Vision (WACV)*, 2011, pp. 606–613. pages 28, 92

[60] S. Dolgov and D. Savostyanov, "Alternating minimal energy methods for linear systems in higher dimensions," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, pp. A2248–A2271, 2014. pages 37, 38

[61] I. V. Oseledets and E. 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. pages 42

[62] E. C. M. Maritz and T. Vetrik, "On the partition dimension of directed circulant graphs," *Journal of Discrete Mathematical Sciences and Cryptography*, pp. 1–10, 2021. pages 47

[63] B. Scalzo, L. Stanković, M. Daković, A. G. Constantinides, and D. P. Mandic, "A class of doubly stochastic shift operators for random graph signals and their boundedness," *Neural Networks*, vol. 158, pp. 83–88, 2023. pages 48

[64] R. Orús, "Tensor networks for complex quantum systems," *Nature Reviews Physics*, vol. 1, no. 9, pp. 538–550, 2019. pages 52, 148

[65] F. Pan, P. Zhou, S. Li, and P. Zhang, "Contracting arbitrary tensor networks: General approximate algorithm and applications in graphical models and quantum circuit simulations," *Physical Review Letters*, vol. 125, no. 6, p. 060503, 2020. pages 52, 148

[66] Y. Zniyed, R. Boyer, A. De Almeida, and G. Favier, "A TT-based hierarchical framework for decomposing high-order tensors," *SIAM Journal on Scientific Computing*, vol. 42, no. 2, pp. A822–A848, 2020. pages 56

[67] I. Kisil, G. G. Calvi, B. Scalzo Dees, and D. P. Mandic, "HOTTBOX: Higher Order Tensor ToolBOX for the Analysis of Multi-way Data," *Journal of Machine Learning Research (JMLR)*, p. tba, 2021. pages 58

[68] H. Johan, "Tensor rank is NP-complete," *Journal of Algorithms*, vol. 4, no. 11, pp. 644–654, 1990. pages 59

[69] D. P. Mandic and J. Chambers, *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. John Wiley & Sons, Inc., 2001. pages 61, 63, 122, 123, 155

[70] Z. He, S. Gao, L. Xiao, D. Liu, H. He, and D. Barber, "Wider and deeper, cheaper and faster: Tensorized LSTMS for sequence learning," *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 1–11, 2017. pages 61

[71] V. Khrulkov, A. Novikov, and I. Oseledets, "Expressive power of recurrent neural networks," *arXiv preprint arXiv:1711.00811*, 2017. pages 61

[72] Y. D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv:1511.06530*, 2015. pages 62

[73] Z. Zhong, F. Wei, Z. Lin, and C. Zhang, "ADA-Tucker: Compressing deep neural networks via adaptive dimension adjustment Tucker decomposition," *Neural Networks*, vol. 110, pp. 104–115, 2019. pages 62

[74] G. G. Calvi, A. Moniri, M. Mahfouz, Q. Zhao, and D. P. Mandic, "Compression and interpretability of deep neural networks via tucker tensor layer: From first principles to tensor valued back-propagation," *arXiv preprint arXiv:1903.06133*, 2019. pages 62, 66, 93, 122, 149

[75] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014. pages 62, 93

[76] R. Yu, S. Zheng, A. Anandkumar, and Y. Yue, "Long-term forecasting using tensor-train RNNs," *Arxiv*, 2017. pages 62, 123

[77] A. Tjandra, S. Sakti, and S. Nakamura, "Compressing recurrent neural network with tensor train," in *Proceedings of 2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 4451–4458. pages 62

[78] M. L. de Prado, *Machine Learning for Asset Managers*. Cambridge University Press, 2020. pages 62

[79] ——, *Advances in Financial Machine Learning*. John Wiley & Sons, 2018. pages 62

[80] B. S. Dees, "Analysing global fixed income markets with tensors," *arXiv preprint arXiv:1908.02101*, 2019. pages 62

[81] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Using deep learning to detect price change indications in financial markets," in *Proceedings of 25th European Signal Processing Conference (EUSIPCO)*. IEEE, 2017, pp. 2511–2515. pages 62

[82] D. T. Tran, A. Iosifidis, J. Kanniainen, and M. Gabbouj, "Temporal attention-augmented bilinear network for financial time-series data analysis," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 5, pp. 1407–1418, 2018. pages 62

[83] D. T. Tran, M. Magris, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Tensor representation in high-frequency financial data for price change prediction," in *Proceedings of IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017, pp. 1–7. pages 62, 91

[84] M. Dixon, "Sequence classification of the limit order book using recurrent neural networks," *Journal of computational science*, vol. 24, pp. 277–286, 2018. pages 62

[85] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Using deep learning for price prediction by exploiting stationary limit order book features," *Applied Soft Computing*, vol. 93, p. 106401, 2020. pages 62

[86] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," *arXiv preprint arXiv:1512.05287*, 2015. pages 62

[87] A. B. Dieng, R. Ranganath, J. Altosaar, and D. Blei, "Noisin: Unbiased regularization for recurrent neural networks," in *Proceedings of International Conference on Machine Learning*. PMLR, 2018, pp. 1252–1261. pages 62

[88] G. G. Calvi, V. Lucic, and D. P. Mandic, "Support tensor machine for financial forecasting," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 8152–8156. pages 66

[89] O. Kuz'kin, "Graph theory methods in analysing commuting networks of municipal electric transport," *Eastern-European Journal of Enterprise Technologies*, vol. 2, no. 4, pp. 19–25, 2016. pages 70

[90] S. Derrible and C. Kennedy, "Network analysis of world subway systems using updated graph theory," *Transportation Research Record*, vol. 2112, no. 1, pp. 17–25, 2009. pages 70

[91] E. Badger, "The evolution of urban planning in 10 diagrams," *City Lab. com*, 2012. pages 71

[92] D. Gattuso and E. Miriello, "Compared analysis of metro networks supported by graph theory," *Networks and Spatial Economics*, vol. 5, no. 4, pp. 395–414, 2005. pages 71

[93] S. Stoilova and V. Stoev, "An application of the graph theory which examines the metro networks," *Transport Problems*, vol. 10, 2015. pages 71

[94] S. Guze, "Graph theory approach to transportation systems design and optimization," *TransNav: International Journal on Marine Navigation and Safety of Sea Transportation*, vol. 8, 2014. pages 71

[95] B. Boulmakoul, Z. Besri, L. Karim, A. Boulmakoul, and A. Lbath, "Combinatorial connectivity and spectral graph analytics for urban public transportation system," *Transportation Research Procedia*, vol. 27, pp. 1154–1162, 2017. pages 71

[96] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, pp. 35–41, 1977. pages 72

[97] M. Anop, "Using graph model to analyze the topological vulnerability of transport infrastructure." in *DOOR (Supplement)*, 2016, pp. 358–366. pages 73

[98] T. Watanabe and A. Nakamura, "Edge-connectivity augmentation problems," *Journal of Computer and System Sciences*, vol. 35, no. 1, pp. 96–144, 1987. pages 73

[99] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3558–3565. pages 80, 81

[100] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems (NIPS)*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, 2018. pages 83, 84

[101] H. Gao and S. Ji, "Graph U-nets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 4948–4960, 2022. pages 83

[102] B. Knyazev, G. W. Taylor, and M. Amer, "Understanding attention and generalization in graph neural networks," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, vol. 32, 2019. pages 84

[103] E. Ranjan, S. Sanyal, and P. Talukdar, "ASAP: Adaptive structure aware pooling for learning hierarchical graph representations," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5470–5477, 04 2020. pages 84

[104] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *Proceedings of the 37th International Conference on Machine Learning ICML*, vol. 119, 2020, pp. 874–883. pages 84

[105] A. H. Khasahmadi, K. Hassani, P. Moradi, L. Lee, and Q. Morris, "Memory-based graph networks," 2020. pages 84

[106] D. Grattarola, D. Zambon, F. M. Bianchi, and C. Alippi, "Understanding pooling in graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11, 2022. pages 84, 86

[107] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007. pages 84

[108] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Hierarchical representation learning in graph neural networks with node decimation pooling," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 2195–2207, 2020. pages 84

[109] E. Noutahi, D. Beaini, J. Horwood, S. Giguère, and P. Tossou, "Towards interpretable sparse graph representation learning with laplacian pooling," 2019. pages 84

[110] L. Stankovic, D. P. Mandic, M. Dakovic, I. Kisil, E. Sejdic, and A. G. Constantinides, "Understanding the basis of graph signal processing via an intuitive example-driven approach," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 133–145, 2019. pages 84, 85, 92

[111] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, 1994, p. 359–370. pages 85

[112] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, p. 561–580, 2007. pages 85

[113] P. Mota, "Normality assumption for the log-return of the stock prices," *Discussiones Mathematicae Probability and Statistics*, vol. 32, no. 1-2, pp. 47–58, 2012. pages 85

[114] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008. pages 85

[115] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of artificial intelligence research*, vol. 11, pp. 169–198, 1999. pages 87

[116] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, pp. 123–140, 1996. pages 87

[117] ——, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001. pages 87

[118] P. Goyal, S. Raja, D. Huang, S. R. Chhetri, A. Canedo, A. Mondal, J. Shree, and C. Jawahar, "Graph representation ensemble learning," in *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2020, pp. 24–31. pages 87

[119] X. Hou, P. Qi, G. Wang, R. Ying, J. Huang, X. He, and B. Zhou, "Graph ensemble learning over multiple dependency trees for aspect-level sentiment classification," 2021. pages 87

[120] J. Wang, S. Zhang, Y. Xiao, and R. Song, "A review on graph neural network methods in financial applications," *arXiv preprint arXiv:2111.15367*, 2021. pages 92

[121] J. Liu, C. Zhu, Z. Long, and Y. Liu, "Tensor regression," *Foundations and Trends in Machine Learning*, vol. 14, no. 4, pp. 379–565, 2021. pages 93

[122] R. Guhaniyogi, S. Qamar, and D. B. Dunson, "Bayesian tensor regression," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 2733–2763, 2017. pages 93

[123] S. E. Sofuoglu and S. Aviyente, "Graph regularized tensor train decomposition," *arXiv preprint arXiv:1911.01591*, 2019. pages 93, 149

[124] Q. Ge, G. Gao, W. Shao, L. Wang, and F. Wu, "Graph regularized Bayesian tensor factorization based on Kronecker-decomposable dictionary," *Computers & Electrical Engineering*, vol. 90, p. 106968, 2021. pages 93, 149

[125] Y. Qiu, G. Zhou, Y. Wang, Y. Zhang, and S. Xie, "A generalized graph regularized non-negative Tucker decomposition framework for tensor data representation," *IEEE Transactions on Cybernetics*, vol. 52, no. 1, pp. 594–607, 2022. pages 93, 149

[126] E. Stoudenmire and D. J. Schwab, "Supervised learning with quantum-inspired tensor networks," in *Proceedings of advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 4799–4807. pages 93, 149

[127] A. Haliassos, K. Konstantinidis, and D. P. Mandic, "Supervised learning for nonsequential data: A canonical polyadic decomposition approach," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2021. pages 93

[128] W. Wang, Y. Sun, B. Eriksson, W. Wang, and V. Aggarwal, "Wide compression: Tensor ring nets," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 9329–9338. pages 93

[129] P. Comon, X. Luciani, and A. L. De Almeida, "Tensor decompositions, alternating least squares and other tales," *Journal of Chemometrics: A Journal of the Chemometrics Society*, vol. 23, no. 7-8, pp. 393–405, 2009. pages 96

[130] S. Zhang, B. Guo, A. Dong, J. He, Z. Xu, and S. X. Chen, "Cautionary tales on air-quality improvement in beijing," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 473, no. 2205, p. 20170457, 2017. pages 103, 127

[131] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. pages 109

[132] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 1416–1424. pages 109, 110

[133] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 1024–1034. pages 110

[134] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017, pp. 1–12. pages 110, 137

[135] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proceedings of the International conference on machine learning (ICLR)*. PMLR, 2016, pp. 2014–2023. pages 110

[136] L. Stankovic and D. P. Mandic, "Convolutional neural networks demystified: A matched filtering perspective-based tutorial," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2023. pages 111

[137] Q. Li, Z. Han, and X. M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the AAAI conference on artificial intelligence*, 2018. pages 119

[138] K. M. Borgwardt, C. S. Ong, S. Schonauer, S. V. N. Vishwanathan, A. J. Smola, and H. P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, pp. i47–i56, 2005. pages 120

[139] X. Di, P. Yu, R. Bu, and M. Sun, "Mutual information maximization in graph neural networks," in *Proceedings of 2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–7. pages 120

[140] D. Q. Nguyen, T. D. Nguyen, and D. Phung, "Universal graph transformer self-attention networks," in *Companion Proceedings of the Web Conference 2022*, 2022, pp. 193–196. pages 120

[141] A. Nouranizadeh, M. Matinkia, M. Rahmati, and R. Safabakhsh, "Maximum entropy weighted independent set pooling for graph neural networks," *arXiv preprint arXiv:2107.01410*, 2021. pages 120

[142] V. Saurabh and Z. Zhi-Li, "Learning universal graph neural network embeddings with aid of transfer learning," *arXiv preprint arXiv:1909.10086*, 2019. pages 120

[143] Z. Zhang, J. Bu, M. Ester, J. Zhang, C. Yao, Z. Yu, and C. Wang, "Hierarchical graph pooling with structure learning," *arXiv preprint arXiv:1911.05954*, 2019. pages 120

[144] Y. Khalifa, D. P. Mandic, and E. Sejdic, "The role of hidden Markov models and recurrent neural networks in event detection and localization for biomedical signals: Theory and application," *Information Fusion, in print*, 2020. pages 123

[145] R. Rohde, R. A. Muller, R. Jacobsen, E. Muller, S. Perlmutter, A. Rosenfeld, J. Wurtele, D. Groom, and C. Wickham, "A new estimate of the average earth surface land temperature spanning 1753 to 2011," *Geoinfor Geostat*, vol. 7, p. 2, 2013. pages 127

[146] "Reports for the UK House Price Index (UK HPI) for England, Scotland, Wales and Northern Ireland." https://www.gov.uk/government/collections/uk-house-price-index-reports. pages 127

[147] F. Palumbo, C. Gallicchio, R. Pucci, and A. Micheli, "Human activity recognition using multisensor data fusion based on reservoir computing," *Journal of Ambient Intelligence and Smart Environments*, vol. 8, no. 2, pp. 87–107, 2016. pages 127

[148] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2017, p. 5998–6008. pages 136, 138, 154

[149] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *OpenAI*, 2018. pages 136

[150] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018. pages 136

[151] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser, "Universal transformers," *arXiv preprint arXiv:1807.03819*, 2018. pages 136

[152] X. Ma, P. Zhang, S. Zhang, N. Duan, Y. Hou, M. Zhou, and D. Song, "A tensorized transformer for language modeling," *Proceedings of Advances in Neural Information Processing Systems*, vol. 32, pp. 2232–2242, 2019. pages 136, 137, 145

[153] O. Hrinchuk, V. Khrulkov, L. Mirvakhabova, E. Orlova, and I. Oseledets, "Tensorized embedding layers," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, 2020, pp. 4847–4860. pages 137

[154] S. Zhang, P. Zhang, X. Ma, J. Wei, Q. Liu *et al.*, "Tensorcoder: Dimension-wise attention via tensor representation for natural language modeling," *arXiv preprint arXiv:2008.01547*, 2020. pages 137, 145

[155] L. De Lathauwer, "Decompositions of a higher-order tensor in block terms—part ii: Definitions and uniqueness," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 1033–1066, 2008. pages 137

[156] H. Chang, Y. Rong, T. Xu, W. Huang, S. Sojoudi, J. Huang, and W. Zhu, "Spectral graph attention network," *arXiv preprint arXiv:2003.07450*, 2020. pages 137

[157] G. Wang, R. Ying, J. Huang, and J. Leskovec, "Direct multi-hop attention based graph neural network," *arXiv preprint arXiv:2009.14332*, 2020. pages 137

[158] W. Sun, S. Chen, L. Huang, H. C. So, and M. Xie, "Deep convolutional neural network compression via coupled tensor decomposition," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 3, pp. 603–616, 2020. pages 149

[159] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," *Advances in Neural Information Processing Systems (NIPS)*, vol. 30, 2017. pages 149

[160] X. Geng, Y. Li, L. Wang, L. Zhang, Q. Yang, J. Ye, and Y. Liu, "Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 3656–3663. pages 149

[161] I. Wickramasingha, A. Elrewainy, M. Sobhy, and S. S. Sherif, "Tensor least angle regression for sparse representations of multidimensional signals," *Neural Computation*, vol. 32, no. 9, pp. 1697–1732, 2020. pages 149

[162] Y. L. Xu, K. Konstantinidis, and D. P. Mandic, "Graph-regularized tensor regression: A domain-aware framework for interpretable multi-way financial modelling," *arXiv preprint arXiv:2211.05581*, 2022. pages 149

[163] K. Wimalawarne, R. Tomioka, and M. Sugiyama, "Theoretical and Experimental Analyses of Tensor-Based Regression and Classification," *Neural Computation*, vol. 28, no. 4, pp. 686–715, 04 2016. pages 149

[164] A. Novikov, M. Trofimov, and I. Oseledets, "Exponential machines," *Bulletin of the Polish Academy of Sciences: Technical Sciences*, pp. 789–797, 2018. pages 149

[165] A. S. Bhatia, M. K. Saggi, A. Kumar, and S. Jain, "Matrix Product State–Based Quantum Classifier," *Neural Computation*, vol. 31, no. 7, pp. 1499–1517, 07 2019. pages 149

[166] C. Chen, K. Petty, A. Skabardonis, P. Varaiya, and Z. Jia, "Freeway performance measurement system: mining loop detector data," *Transportation Research Record*, vol. 1748, no. 1, pp. 96–102, 2001. pages 158

[167] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *arXiv preprint arXiv:1707.01926*, 2017. pages 158, 159

[168] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, vol. 27, 2014. pages 159

[169] M. Villani, J. Lockhart, and D. Magazzeni, "Feature importance for time series data: Improving kernelshap," *arXiv preprint arXiv:2210.02176*, 2022. pages 162

[170] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R Ruiz, J. Schrittwieser, and G. Swirszcz, "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, vol. 610, no. 7930, pp. 47–53, 2022. pages 164