Chapter 6: Tensor Computation Applications

This chapter presents higher-level application domains for the various methods discussed in the previous chapters to identify trends and challenges. Tensor computing and analysis have been successful in various application domains, such as social network analysis, brain data analysis, web mining, information retrieval, healthcare analytics, signal processing, machine learning, and predominantly psychometrics and computational chemistry. In this chapter, we will select several applications to investigate how the various tensor analytics models have been applied, identifying some preprocessing steps, details of application methods and programming environment used, results' analysis and performance evaluation compared with matrix methods.

It is impossible to be comprehensive or select the best representative research outcome. An attempt to diversify the application domains and methods applied was made. Most models are 3-way models. We previously discussed a 3-way model in chapter three about MicroRNA-Disease Associations. The video application discussed in chapter four was a 4-way model. The BSS application in chapter four was a higher-order tensor. Chapter five discussed various applications of representation learning and feature extraction methods, followed by one application using tensorial methods in Multi-Task Learning. The applications in this chapter will span scientific computation applications, data science using various dataset types and requirements, and deep learning models. We will summarise in an attempt to create a general framework for tensorial methods in machine learning applications.

6.1 Scientific Computing Applications

Since Tensor computing was initially exploited in the psycho- and chemo-metrics communities and was founded by the quantum mechanics & physics community, we will start with one application from each of these three domains. We already covered a bioinformatics application in chapter three, covering the popular areas of scientific computing. We will then move on to data science and machine learning up to deep learning models.

Pieter Kroonenberg is one of the founders of multi-way analysis algorithms, from data preprocessing, model selection, and application problems in psychometrics, and social and behavioural sciences to agriculture, environmental sciences, and chemistry, to results interpretation and their validation and visualisation. His book (Kroonenberg, 2008) offers a rich, easy-to-read explanation of the methods with reduced mathematical notation. The methods covered vary from Tucker and CP tensor decomposition, dataset preprocessing and handling of missing data to core factors interpretation to achieve multi-way PCA, multi-way correspondence analysis, multi-way factor analysis, multi-way clustering, and improve the model by scaling and rotation, and finally, results' validation and analysis of residuals. His website at https://three-mode.leidenuniv.nl/, published the implemented algorithms, their applications, and the datasets. He has the software "3WayPack" that can be downloaded in Windows XP to execute the applications discussed in the book. The datasets are also available on the website to download and experiment with other recent development methods such as Matlab, R, Python, and others. We will review the Happiness dataset and rebuild the model he explained in the book using Python. The survey dataset has a predictor in the first mode as the number of school years a subject finished (4 ordered categories), a second predictor in the second mode as the number of siblings (5 ordered categories), and their dependent variable is the happiness score in the third mode (3 ordered classes). The analysis is a correspondence analysis to identify the interactions between the variables. Data preprocessing included analysis of correlation and analysis of variance. The book evaluated five models, from 1x1x1 tensor to 2x2x2 tensor evaluating the Tucker3 to identify the Residual Sum of Squares SS as SS(Total of Mode A) = SS(Fit of Mode A) + SS(Residual of Mode A), and similarly for the other two modes. Thus, $SS(Residual)_i$ / $SS(Total)_i$ is the proportional or relative lack of fit of the i^{th} level of mode A. This showed that the 2x2x2 model is the best model with a Proportional SS(Fit) of 86% as opposed to smaller models down to 0.69% for the 1x1x1 model. The results were validated using $\mathcal{X}^2$ (chi-square) statistic scores for expected different pair-wise interactions between each predictor and the dependent variable, then the combined interactions of both predictors on the target variable. The extended statistical investigation is interesting to follow to validate the end-to-end results of some AI output. This problem was reproduced in Python in the motivating problem of section 3.1 in chapter three accompanying source code "multi-wayExamples.ipynb".

For the chemo-metric application, we will review a chromatography application using a tensor decomposition example that is recently published. In computational chemistry, chromatography separates a compound into its individual components for separation or identification objectives. Traditionally it can be solved by ICA methods such as in the BSS problem introduced in chapter two. The dataset used in the analysis is multi-way by nature—for example, different chemicals, solvents and stationary phases. A two-

dimensional gas chromatography (GC×GC) is usually accompanied by a time-of-flight mass spectrometer (GC×GC-TOFMS) and has applications in food/drug/environmental contaminants/petrochemicals or other compounds safety and quality analysis. The lab process is to inject a solvent of a specific type into a tube containing a compound to separate its molecules. Then measure the Time Of Flight (TOF) of sample molecules on a detector such as an ion mirror/reflector on which ions with higher energy penetrate more deeply inside, extending the time when they are reflected. The detector performs regular measurements at regular intervals. Since the flight times of the separated ions are proportional to the square root of respective mass-to-charge ratio (m/z values), identifying the different molecules' types. An excised region is a vector of length I, where I is the number of acquisitions in the region of interest of a single chromatogram. A multivariate detector spans the second mode of J variables or channels. This is usually performed over several runs. The challenge is that chemical components are free to shift independently along the first and second chromatographic modes between runs and employ different chromatographic modes. (Armstrong et al., 2022) proposed using a fourth-order tensor $\mathcal{X}^{I\times J\times K\times L}$ of I mass spectral acquisitions, J mass-to-charge ratios (mass channels), K modulations, and L samples as multiple samples from the same region of the chromatogram or entire chromatograms.

The authors applied PARAFAC to decompose $\mathcal{X}$ as $\mathcal{X} = F_2(D_l \odot F_1 \odot A)^T$, such that F_2 is as I × R matrix, A is a J × R matrix, F_1 is a K × R matrix, and D_l is an L × R matrix. They employed a tensor unfolding in the first two modes to produce the third order $\mathcal{X}_l{}^{I*J\times K\times L}$ avoids the problem of drift in two modes by artificially reducing the problem to drift along one combined retention mode. Another approach is to form tensors of similar orders by stacking second-dimension retention profiles for an $\mathcal{X}_{kl}{}^{I\times J\times K*L}$, and first-dimension retention times' $\mathcal{X}_{il}{}^{K\times J\times I*L}$. Each model has useful properties, and using both models simultaneously to minimise (using ALS), the 4-way tensor reconstruction from the 3-way tensors arrives at optimal convergence at a retention mode at which the highest resolution between closely co-eluting (extraction) chemical factors. The approach is called the PARAFAC2×2 algorithm and is shown to achieve good results in synthetic and real data from a metabolomics study and is extensible to higher-order chromatographic separations. They published their Matlab code at https://github.com/mdarmstr/parafac2x2. Earlier simpler Parafac on three-way tensor decomposition method compared to PCA for chromatography are explained in chapter ten in (Smilde et al., 2004). A sample three-way chromatography dataset can be downloaded from https://three-mode.leidenuniv.nl/data/chromatographyinfo.htm.

The physics community and the mathematicians found everything about tensors, groups, representation and transformation. It is worth including some physical applications to tensor computing methods, besides the recurring example of Inertia matrix and SO(3)

matrices and their applications in computer graphics and robotic motion control. Later in the neural networks section of this chapter, an application of quantum mechanics will be reviewed. In chapter four, some applications of the TT decomposition method were discussed, including the density matrix renormalisation group (DMRG) algorithm optimising tensors with variable directions that has applications in solid-state physics for solving eigenvalue problems. Another TT application is the iterative alternating minimal energy (AME) algorithm that has applications in approximating solutions of higher dimension and multi-dimension systems of linear equations Ax=b, on which the Gaussian Elimination method would be intractable. Matrix factorisations have been used to speed up the computation when NxN matrix A is of a large N dimension, such as LU decomposition keeping the computation in $O(N^3)$. Also, Iterative methods such as the generalized minimal residual (GMRES) algorithm based on Krylov subspaces are in $O(N^2)$ complexity class but have problems of orthogonality for dense high dimensional matrices. The authors in (Lantsov, 2021) discuss converting the linear equations matrix to a tensor and applying tensor decomposition methods to reduce the memory requirements and speed up the calculations. Modern radio-frequency (RF) circuits and simulation of electronic circuits in CAD systems are high dimensional. For example, in CAD, electronic circuit design tools require analysis of different types such as static mode analysis (direct currents analysis, DC), small-signal analysis (linear analysis), transient analysis (time domain analysis), non-linear distortion analysis (based on Volterra series), harmonic balance analysis, and several others as identified by the authors. The authors represented the equations high dimensional NxN matrix A with N^2 elements as a low-rank tensor of N-order, using the TT decomposition as a tensor B with 2^{2D} elements, where $D = \log_2 N$.

Similarly, matrix b was converted to the TT format. The conversion used a binary decomposition algorithm that they implemented in Matlab. Solving for x, the authors applied the DMRG algorithm, which was not as reliable as the AME algorithm, but both are computationally more efficient than the GMRES algorithm. DMRG and AME algorithms are implemented in TT Matlab Toolbox and other programming languages, including in Python in https://www.tensors.net/p-dmrg and explained in http://tensornetwork.org/mps/algorithms/dmrg/.

6.2 Data Mining Applications

We will consider any machine learning application, whether working on a text dataset, image or video dataset, semantic web or knowledge graphs, or any other format, as a data mining application. The first few applications are working on text datasets. However, no semantic encoding of the words is applied. Other NLP applications are reviewed while reviewing neural network applications.

The first application of tensors in data mining to review is contributed by Acar et al. (Acar et al. 2005), (Acar et al., 2006), who applied different tensor decompositions to the problem of discussion disentanglement in online public chatrooms on the Internet Relay Chat (IRC), and how social networks evolve. The authors implemented their own bot to collect their dataset and simulated some, containing the text messages with timestamps, nicknames of identities, and timestamps of nicknames quit/leave or kick. The nicknames could belong to the same person, pretending to be of any age and gender. The topics' keywords were semantically analysed, and data distributions were estimated. A similar dataset without the nicknames is found at https://www.tensorflow.org/datasets/catalog/irc_disentanglement. The dataset is multidimensional and noisy.

The authors constructed a tensor T of order three, capturing users, keywords, and time in each mode, respectively, such that where T_{ijk} is user i, sent a number of keyword j, during time slot k. Then Tucker1 and Tucker3 were applied to identify user groups, which are set of users sharing a maximal keyword set in a given time period. This is achieved by an indeterministic c-means clustering algorithm running 100 times, which returns multiple memberships for each data point. The authors added different levels of Gaussian noise. Methods such as SVD as a 2-way dimensionality reduction method were applied in this paper on a matrix of users and keywords (UK) and another matrix of users and time stamps (UT). SVD clustered user groups with common keywords independently, and another cluster using the second matrix of users chatting at the same period. By tracing the resulting groups from all experiments at three different noise ratios, the multi-way structure connecting the three modes was more accurately captured by the multi-way tensor decomposition methods (Tucker1 and Tucker3) than by the SVD on UT method, while it failed by SVD on UK method. Then they defined a 4-way Tensor to add the IRC server as the fourth mode to identify the computational efficiencies of tensor higher-order representations and decomposition methods compared to the corresponding pair-wise approaches.

The main lesson to learn from the previous paper is the data collection for tensor decomposition. Most datasets available in the public domain, such as Kaggle, Google Datasets, and UCI, are in matrix form. Data reformatting in matrix form as presented in the "tensorisation.ipynb" and "multi-wayExamples.ipynb" or other creative methods need to be planned for every experiment.

In text analysis applications, (Bader et al., 2008) used CP for automatic conversation detection in the Enron Email dataset over time using an m term x n author x q month 3-way tensor $\mathcal{X}^{\mathrm{m\,x\,n\,x\,q}}$. This dataset is available at https://www.cs.cmu.edu/~enron/. Based on previous literature on the analysis of this dataset, the authors selected an interesting subset of this dataset that makes a tensor $\mathcal{X}^{69157 \times 197 \times 12}$ with 1,042,202 non-zeros entries scaled to their weighted frequency. The authors used a non-negative tensor to avoid

subtractive basis vectors and encoding interactions that are found in PCA-like methods. They applied Parafac decomposition using ALS such that $\min_{A,B,C} \|\mathcal{X} - \sum_{l-1}^{r} A_l \circ B_l \circ C_l\|^2$. The decomposed tensor factors were A^{mxr} containing highest scores for terms/topics, B^{nxr} containing highest scores for authors, and C^{qxr}, containing the highest scores for these topics over time; they chose a 12-month duration. The rank r was chosen to retrieve a specific number of topics in the data, setting it to 25. Eight topics out of the 25 were interpretable in the context of other events happening in the same time duration, as compared to the two-way (term-author) NMF method that could not extract these discussions. The decomposition predicted discussion threads and produced charts of previous focused discussions over time.

6.2.1 Knowledge Graphs

The probabilistic graphical models like Bayesian and Markov networks are classical approaches to learning from network and graph data. However, they require defining priors and data distributions and are compute-intensive for high-dimensional datasets (Schlüter, 2014). Tensor decomposition approaches are more accurate in the collective learning of network data and the prediction of new relations in knowledge graphs, with a trade-off between expressivity and computational efficiency than non-tensorial methods. They were first used with probabilistic graphical models, such as the Bayesian Clustered Tensor Factorization (BCTF) as proposed by (Sutskever et al., n.d.), in which CP tensor decomposition was used to analyse network traffic and bibliographic data. The authors of (Nickel et al., 2011) proposed a relational learning approach RESCAL based on the DEDICOM tensor decomposition method with relaxed constraints. They exploited a three-way tensors $\mathcal{X}^{\mathrm{n} \times \mathrm{n} \times \mathrm{m}}$, n entities x n entities x m relationships such that when $\mathcal{X}_{ijk} = 1$, this means entity i has a relationship of type k with entity j. Entities. Domain data is given in form of RDF triplets. A tensor decomposition of the form $\mathcal{X}$as $\mathcal{X}_k = AR_kA^T$, where A is a n×r matrix containing the latent-component representation of the repeated entities in two modes from the domain and R_k is an asymmetric r×r matrix that models the interactions of the entites in the k-th predicate. This is very close to a relaxed DEDICOM or equivalently, an asymmetric extension of IDIOSCAL tensor decomposition method. RESCAL is an efficient minimisation algorithm with regularsation term, based on the ASALSAN (Alternating Simultaneous Approximation, Least Squares, and Newton") variant of the ALS approach. The computed low rank representation of the domain data, is used in preduction of a link as $\widehat{\mathcal{X}}_{ijk} > \theta$, for some threshold θ. The collective classification can be performed by slicing the low rank reconstructed tensor for a given class relationship, or actually only reconstructing only the relevant slice. Also, Link-based clustering of entities can be performed using a similarity measure between entities based upon their similarity across multiple relations. The authors compared the performance of RESCAL compared to standard tensor factorizations such as s

CP and DEDICOM, and relational learning algorithms such as statistical unit node set (SUNS) and the aggregated SUNs+AG. The conducted various experiments on various datasets such as Cora dataset (https://relational.fit.cvut.cz/dataset/CORA), Kinships in Australian Tribes, Nations (clusters of countries, clusters of interactions between countries, and clusters of country features) and UMLS (Unified Medical Language System) that can be found at (https://github.com/ZhenfengLei/KGDatasets). They showed that the results of RESCAL and DEDICOM outperform both CP and SUNS on all datasets.

The authors repeated the experiment (Nickel et al., 2012) using the noisy, large and high-dimensional Semantic Web's Linked Open Data (LOD) cloud that contains millions of entities, hundreds of relations and billions of known facts, found at https://lod-cloud.net/ and explained at https://www.ontotext.com/knowledgehub/fundamentals/linked-data-linked-open-data/. When writing this paper, it connected 300 datasets; now, it connects 1,255 datasets with 16,174 links. They employed the YAGO 2 ontology (that can be found at https://yago-knowledge.org/) that at the time contained $4.3x10^{14}$ possible triplets to improve the learning by factorising a large knowledge base using an extended RESCAL tensor factorisation method to include in the factorisation the attributes of the entities, i.e. the literal values in LOD. Due to the large dataset content, the authors used a map-reduce parallel programming paradigm to employ distributed computing nodes to speed up the processing. RDFs are based on ontologies that contain the T-Box (terminological component), while the instance data are contained in A-Box (assertion component). The formed tensor $\mathcal{X}^{\mathrm{n}\times\mathrm{n}\times\mathrm{m}}$ represent both T-Box and A-Box simultaneously, making ontology a soft constraint, and the model is data-centric and ontology-driven. The model was tested for ranking and unknown triplets prediction and retrieval of similar entities using the latent factors in the decomposed core matrix A. Another application is the proposal of new ontologies' terms from the data to knowledge database engineers as decision support systems to evolve an ontology by grouping/clustering instances. Out of 87 predicates relation in YAGO 2 at the time, 38 were used to form a sparse tensor $\mathcal{X}^{3000417\times3000417\times38}$ with 41 million entries and a sparse attribute matrix $D^{3000417\times1138407}$with 35.4 million entries. Of the 4.3x1014 possible triplets in YAGO 2, only 4x107 non-zero entries were available. The authors compared the RESCAL to other tensor factorisation and classical relational learning methods, showing that RESCAL is more efficient in predicting RDF triplets and other machine learning tasks.

The authors of (Padia et al., 2016) further extended the RESCAL tensor decomposition method to RESCAL+ approach by adding a similarity matrix to the minimisation algorithm to force slices of the relational tensor to decrease their differences between one another to achieve unequal contribution of the slices. They tested using the DBpedia-Person dataset found at https://www.dbpedia.org/. They compared their link prediction performance

against the original RESCAL method and its non-negative variant NN-RES and showed that their method achieves higher AUC scores and diagonal confusion matrix scores.

The RESCAL method is implemented in Python in https://github.com/mnick/rescal.py and in the scikit-tensor library along with CP, Tucker, DEDICOM, and INDSCAL found at scikit-tensor library.

The authors of (Lacroix et al., 2020) applied CP decomposition to predict dynamic knowledge graph links. They created a 4-mode tensor of subject, predicate, object, and time. They proposed a new dataset for temporal knowledge graphs parsing Wikipedia. They compared their model to other models on their proposed dataset and other datasets to show that their results are promising. They published their code at https://github.com/facebookresearch/tkbc.

A generative model using Bayesian Tucker decomposition is proposed by (Castellana and Bacciu, 2019) that is suitable for tree-structured data. The Markov model is an expressive model that grows in size and becomes intractable for practical problems. Tensor factorisation enables the model to be a non-parametric Bayesian model as well.

6.2.2 Computer Vision

Eigenfaces is an algorithm for face recognition. It models several images for the same person as an unfolded vector of each image that is stacked in a matrix. Then, Eigen decomposition using principal component analysis (PCA) is applied to reduce the dimensionality of the images to use only the uncorrelated variables. The result is the eigenface with the smallest Euclidian distance to which the person resembles the most (Turk and Pentland, 1991). Their code is implemented at https://github.com/svetlana-topalova/eigenfaces. The spatial image content is represented in XY matrices for P people, so each matrix represents only one person. Capturing pair-wise variance build models that work best when only a single factor varies; in eigenfaces, it is the person's identity but loses efficiency when compound factors vary, such as poses, viewpoint, and others. The authors of (M. Alex O. Vasilescu and Terzopoulos, 2002) pioneered the use of Tucker decompositions in computer vision to disentangle the multiple factors an image is composed of, such as scene structure, different facial geometries (people), expressions, head poses, lighting conditions, and imaging. They contributed TensorFaces, representing the tensor decompositions' cores as a set of facial components. They applied a HOSVD on a facial images (512 × 352 decimated by a factor of 3 and cropped yielding 7943 pixels) dataset of 28 people x 5 poses x 3 illumination condition x 3 facial expressions x 7943 pixels. This created a 5-mode tensor $\mathcal{D}^{28\times5\times3\times3\times7943}$, that is decomposed by HOSVD to: $\mathcal{D} = \mathcal{Z} \times_1 U_{people} \times_2 U_{views} \times_3 U_{illumnation} \times_4 U_{expression} \times_5 U_{pixels}$, such that $U_{people} \in \mathbb{R}^{28\times28}$spans the space of people parameters, $U_{views} \in \mathbb{R}^{5\times5}$spans the space of viewpoint parameters and so forth for the remaining factor matrices. Each Factor matrix is computed as $U_n = \mathcal{D}_{(n)}V_n\Sigma^+$, where $\mathcal{D}_{(n)}$is mode n flattening of $\mathcal{D}$, and computing SVD on $\mathcal{D}_{(n)}$ to

have the right matrix V and the singular values in the diagonal matrix Σ, considering U_n to be the left matrix of the SVD. This method generalises the eigenfaces method as factor matrix U_{pixels} is the eigenimage. Solving for the core matrix $\mathcal{Z}$ calculates the interactions of all modes considered in this experiment, as explained in the Tucker Decomposition in chapter three. PCA is a change of basis mapping, such as in eigenfaces. Each person will be represented parsimoniously using the different PCA coordinates. Similarly, $\mathcal{Z}$ and all U_n for all the modes considered, can be thought of as a change of basis/coordinates mapping functions to retrieve slices of the Tensor $\mathcal{D}$ that abstract a person as poses change, or as illumination changes, or as expression changes. This is achieved by reducing the intra-class or intra-person interactions so as not to confuse people together by maximising the inter-class or inter-person interactions of the changing conditions considered. You can refer to the original paper for images abstracting the variation of illumination, poses and expressions independently.

A sample Tensorfaces implementation using only 3-mode tensors using the Tensorly Python package can be found at https://github.com/tensorly/Proceedings_IEEE_companion_notebooks/blob/master/tensorfaces.ipynb. They formed a 3-way tensor $\mathcal{X}^{38\times50\times2900}$ of 38 people x 50 illuminations condition x and 2900 pixels from the Yale Face Database found at http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html.

Additional modes such as camera angle and more can also be incorporated using the tensors machinery. The authors compared TensorFaces to eigenFaces and PCA approaches to facial recognition and found the accuracy of TensorFaces is significantly more accurate than standard PCA techniques and can separate each factor's interactions while considering compound interactions as well (M.A.O. Vasilescu and Terzopoulos, 2002). They also worked on reducing the dimensionality (working on subspaces) of every core matrix to achieve compression and remove irrelevant effects, such as illumination, shadows and highlights were removed, while retaining key facial features (Vasilescu and Terzopoulos, 2003).

The authors of (Lehky et al., 2020) expanded the analysis of the TensorFaces representations to measure the reconstruction errors at different complexities levels (ranks). They used synthesized faces generated by FaceGen software (https://facegen.com/), which they modelled in a 4-mode tensor, keeping the spatial image width and height as the first two modes (Tensorfaces was vectorising the image as 1D in one mode), a colour dimension, and the different individuals' dimension. The training dataset contained 128 faces, and the test set contained 40 faces, all equally from males/females and from four different races at the same rotation, pose and camera angle. Their aim was to attempt a reconstruction of these faces at different levels of complexity and measure the reconstruction error to decide the best level of complexity. They experimented with various measures of complexity and errors using TensorFaces, and other pair-wise methods such as PCA and ICA. They showed that low-complexity representations are better for novel faces (not seen in the training dataset) than high complexity representations. This means that tensorial methods achieve regularization as well as multi-way interactions.

Vasilescu in (Vasilescu, 2002) has also applied the Tucker decomposition to human motion as a composite of multiple actions. The author had multiple aims, to extract a human movement signature as a subset of actions (analysis), to resynthesize new motions from the learned ones (synthesis), and recognise a specific person or action (recognition). The author defined a tensor $\mathcal{D} \in \mathbb{R}^{\mathrm{N}\times\mathrm{M}\times\mathrm{T}}$, where N is the number of people, M is the number of action classes, and T is the number of joint angle time samples. The author applied the same N-mode SVD applied in TensorFaces, $\mathcal{D} = \mathcal{Z} \times_1 P \times_2 A \times_3 J$. The people core matrix P has N person-specific rows containing the human motion signatures. The action core matrix A has M action-specific rows encoding action invariances across people. The last joint angle matrix J has T joint angles rows, the eigenmotions typically computed by PCA. The analysis is achieved by a change of basis to capture a person-associated motion by $\mathcal{C} = \mathcal{Z} \times_1 P \times_3 J$, and the change of basis to capture an action-associated motion is achieved by $\mathcal{B} = \mathcal{Z} \times_2 A \times_3 J$. The synthesis is achieved by the knowledge of the core tensor $\mathcal{Z}$ capturing all multi-way interactions, core matrix A generalising the actions, and core matrix J generalising the joint angles. To synthesize an action of a new person not seen before is a Tensor completion or a regression problem to predict $\mathcal{D}_{p,a}$ of a new person p doing action a, as $\mathcal{D}_{p,a} = \mathcal{B}_a \times_1 p^T$, where $\mathcal{B}_a = \mathcal{Z} \times_2 a_a^T \times_3 J$ for the specific action a. If the aim is to synthesie a motion for a new individual, then $p^T = d_a^T \mathcal{B}_a^{-1}$, where d_a^T is the flattened tensor in the people mode, and choosing the specific action Transpose, and the complete set of motions for the new individual is $\mathcal{D}_p = \mathcal{B} \times_1 p^T$. If the aim is to synthesize a new action for a known person that we have other actions recorded for, the new action $a^T = d_p^T \mathcal{C}_p^{-1}$, then synthesizing this action for all people in the database is $\mathcal{D}_a = \mathcal{C} \times_2 a^T$. This makes this tensor decomposition a generative model. In machine learning, discriminative models, like most classification models, learn from a prelabelled dataset how to discriminate the different classes. While a generative model, specifically the Bayesian probabilistic model, understands how samples are generated and can create new data instances from the known ones. Also, the recognition task is achieved in this tensor decomposition by identifying a person from action parameters as the projection $p = \mathcal{B}_a^{-T} d$. Similarly identifying a person's specific action is the projection $a = \mathcal{C}_p^{-T} d$. In both cases, the neareset neighbour algorithm returns the nearest person or action in the learned motion data d.

The author experimented using a Human limb motion dataset collected using a VICON system that employs four video cameras detecting the 3D position of 18 infrared markers placed on each person's legs. Six persons moved in different actions across a 12-meter walkway, such that two cameras could observe each marker during the three different motions (walking, ascending/descending stairs). A similar dataset is collected by (Wang et al., 2021), and others are found at https://www.handcorpus.org/?tag=humanmotiondata. Vasilescu validated the ability of the tensor decomposition model to synthesize new motions for people and identify people from motion or actions from recorded motions.

6.3 Deep Neural Networks Compression

Chapter five, all previous applications in this chapter and other chapters, required hand-tailored feature engineering to define a suitable feature extraction of best representation based on some criteria. The criteria can be to achieve dimensionality reduction, model compression, model regularisation, or data representation invariance to some transformations such as translation, rotation, scale and others. DNN, since 2006 have been evolving to solve many problems using linear algebra concepts hierarchically to model non-linear relationships without hand-crafting the representation of the features or the model parameters. Neural Networks are considered function approximators, starting from the simplest regression model using one input layer and one output layer. Adding more layers learns more complex data representation and functions. These approaches are labelled feature engineering and are known to enhance the performance of any machine learning algorithm. Features transformations include Fourier Transforms and their variants, such as STFT, MFCC, or wavelets transform and their variants, and others such as SOFT, HOD for image, audio and video datasets, BoW and TF-IDF for text datasets.

While neural networks implicitly learn the representation, they achieve compression and regularisation by approximations that are achieved through the layers by neglecting irrelevant contents, such as the pooling layers in CNNs applying different pooling functions. This process studies the structural data interactions, forming an embedding representing the data that can be used for the given machine learning task at the output layer. The depth of the neural networks adds more parameters to estimate, causing the curse of dimensionality in big data analytics. It was observed that the weights within a layer in CNN can be estimated by a 5% subset of its parameters, indicating the DL models are over-parameterised (Denil et al., 2014).

The increase in parameters from the increased number of layers created the need for tensor structures and algorithms to reduce the computational complexity without losing multiple interactions by the overly-simplistic pair-wise interactions. A direction of improvements for DNN performance was labelled compressive DNN, which aims to reduce hardware requirements and enable running in embedded devices such as phones and IoT (Wu et al., 2020).

The simplest method to compress a NN is to add the famous pooling layer that primarily applies the max (to keep only the highest values as the bright colours of an image only) or average or other functions based on the application requirements, on the weights specifying how much spatial reduction in size is required. This is a blind compression method. Another blind compression is the famous dropout layer that randomly sets some neurons to zero; this approach is called pruning. This is useful for regularisation to prevent the network from

overfitting. Other methods work to learn which neurons to drop out in a structured approach, such as (Fan et al., 2019) and (Knodt, 2022).

Other compression approaches, such as Vector quantisation methods, were applied to the CNN parameters and storage requirements. These methods include binarization (1-bit quantization) by turning off neurons with negative values. Another Quantisation method uses lower precision, such as converting floating point types to integer types compressing the network four times and speeding it up 2:4 times. Moreover, scalar quantisation uses k-means to cluster the weights and use representative neurons of each cluster. Also, Product quantisation divides the weights vector space into many disjoint subspaces and quantises them by raising them to different powers and applying k-means on all and storing their cluster indices only. Finally, the residual quantisation performs clustering by k-means and then identifies the residuals to reapply the clustering on them (Gong et al., 2014).

Another NN compression approach is HashedNets (Chen et al., 2015), which uses a hash function to group connection weights in hash buckets. A single parameter for each hash bucket is stored for the connections sharing the bucket. The traditional matrix factorisation methods (low-rank matrices) such as SVD on the weights matrix and only storing the highest singular values corresponding weights. Layer Fusion (Graph Optimization) is another compression approach that minimises computation, memory storage, and bandwidth by combining successive computational graph nodes into a single node for kernel execution.

Also, various tensor factorisation algorithms have been applied to achieve NN compression. This approach requires tensorizing and decomposing the weight matrices into a series of low-rank tensors to reduce redundant weights by using sparse representations. Due to the DL enhancements, both applications of computer vision and NLP applications research advanced through the years. We will start with these two application domains. Then, deep generative models, multi-modal applications and graph neural network (GNN) models will be discussed.

6.3.1 Computer Vision

The first application advances were facilitated by using CNN with deeper layers than ever before. Although the transform to the Fourier Domain was observed to speed up the convolution performance by 200% to achieve higher accuracies in capturing more complex interactions in a dataset, deeper layers have always been the solution, increasing the number of parameters and leading to the need for NN compression. Although, research trends are still mixing up explicit representation encoding by adding more layers, such as (Nair et al., 2020) propose using Fourier Transform Layer in CNN to optimise object detection. The need for model compression is needed in all network models.

The authors of (Novikov et al., 2015) replaced fully connected layers in CNN with a Tensor Train (TT-Layer) that they called TensorNet, which is compatible with the same training algorithm. As explained in the previous chapters, all tensor decomposition approaches capture more multi-way structural interactions in the dataset, making them more expressive than pair-wise matrix factorisation or compression approaches. TT Format is more immune to dimensionality curse and simpler for basic operations (addition and multiplication by a constant, summation and entry-wise tensor products, the sum of all elements and Frobenius norm) than Tucker and Hierarchical Tucker tensor decompositions. They proposed a mapping function between the weights matric and the TT-format and introduced the NN layer with weights stored in TT-format to be a TT-layer, which, when used in any NN, makes it TensorNet. A fully connected layer computes $y = Wx + b$, while a TT-layer converts all to tensors in TT-format $\mathcal{Y}(i_1, \dots, i_d) = \mathcal{G}_1(i_1, j_1) \dots \mathcal{G}_d(i_d, j_d)\mathcal{X}(j_1, \dots, j_d) + \mathcal{B}(i_1, \dots, i_d)$, where $\mathcal{G}_i$are the d core tensors of the TT-format of the original weights matrix, and $\mathcal{Y}$, $\mathcal{X}$ and $\mathcal{B}$ are the d-dimensional tensors formed from the corresponding vectors y, x, and b, respectively. The paper details how the loss function can be performed in the TT-Format. They experimented with MNIST, and CIFAR-10 datasets, comparing a baseline two-layer fully connected network with the two-TT-Layer TensorNet using different ranks (compression factors) to show that they achieved 200,000 times fewer parameters and compressing the size of the whole network by a factor of 7, without compromising the accuracy on TT-ranks all equal to 8. They tested the ImageNet dataset using a different arrangement of layers of TT with various ranks, FC, and MR (matrix rank restricted) on the VGG-16 and VGG-19 networks. All experiments show TensorNet to be the most compressed, most accurate and faster both on CPU and GPU and has the potential to improve the performance of wide (more neurons) and shallow (fewer layers) networks that were known for overfitting. More details about their experimental setup details can be found in their paper, and their Matlab code is published at https://github.com/Bihaqo/TensorNet. TT-layer was implemented by the Tensorly package at https://github.com/tensorly/Proceedings_IEEE_companion_notebooks/blob/master/tt-compression.ipynb/.

The previous application employed TT tensor decomposition on the single layers of the network to efficiently store the dense weight matrices of the fully-connected layers of a VGG network. Conversely, (Calvi et al., 2020) introduce the Tucker Tensor Layer (TTL) as an alternative to the dense weight matrices of neural networks. They also showed how the number of parameters in the neural layer is reduced while deriving a Forward and back-propagation on tensors algorithm that preserves the physical interpretability of Tucker decomposition and provides an insight into the learning process of the layer. Using various compression factors, they also tested on the MNIST, Fashion-MNIST, and CIFAR-10 datasets using the VGG-16 network. For example, they achieved a 66.63% compression with 82.3% accuracy compared to 86.3% accuracy of the uncompressed model.

The Hierarchical Tucker (HT) tensor decomposition method performs better in compressing the weight matrices in Fully Convolutional layers because HT prefers the tensor with balanced dimensions lengths, as shown in (Gabor and Zdunek, 2022). The authors experimented with medium-scale CNNs on the CIFAR-10 dataset and large-scale CNNs, such as VGG-16 and ResNet-50, on the ImageNet dataset. They compared HT-2 to other tensor factorisation and other NN compression approaches to show its competitiveness in the achieved compression without much drop in accuracy. Their code is published at https://github.com/mateuszgabor/ht2.

A hybrid tensor decomposition combining TT and HT is proposed in (Wu et al., 2020). The authors tensorized the input X and the weight matrices W, applied the tensor decomposition on the weights tensor and updated the forward pass WX multiplications for fully connected layers, RNNs and LSTMs. They also tensorized the kernels in CNNs and then explained the tensorized the gradient calculations in the back-propagation step. They compared HT formats to TT-LSTM (Yang et al., 2017) and TR-LSTM (Pan et al., 2018) models on the UCF11 and UCF50 video classification datasets. Then they compared kernel compression in CNNs on the CIFAR-10 dataset and adopted 3D-CNNs to recognize videos on UCF11 and CVRR-HANDS 3D datasets. They show that RNNs/LSTMs in the HT format have higher compression than those in the TT format when compressing weight matrices but with worse accuracy than regular uncompressed RNNs. They also showed that the TT format is more suitable for CNNs, achieving higher accuracy but similar compression to the HT format. Comparing the proposed hybrid tensor decomposition method on CNN models, it has both higher accuracy and higher compression than the TT format and outperforms the uncompressed models in the CVRR dataset.

6.3.2 NLP Application

Bag of words (BoW) and other frequency-based methods such as Term Frequency – Inverse Document Frequency (TF-IDF) of encoding text data were used to capture some semantic context for the data. These failed to capture the different meanings one word could have when placed in different sentences with other words or even in a different order in a sentence with the same words. Initially, semantic understanding of text depended on rule-based creation such as building ontologies and semantic web tools such as Resource Description Framework (RDF). This provided trinary relationships in which many words' meanings and contexts can be captured. These methods can be easily built using a graph database.

(Socher et al., 2013) proposed Recursive Neural Tensor Network (RNTN) that uses a high-order neural network for structured data that leverages a full 3-way tensor for aggregating children's information in binary parse trees within a natural language processing application.

Their method built the Stanford Sentiment Treebank dataset that is available at http://nlp.stanford.edu/sentiment. They build a tree for the word vectors in the embedding matrix $\mathcal{L} \in \mathbb{R}^{d \times |V|}$, where d is d-dimentional word vector, and |V| is the number of words. This binary tree has leaves of the ordered words in the corpus, and learn the parent p_i as hidden vectors that are functions (such as tanh) of the two children that, at the first level, are two given words. The recursive neural networks (RNN) models generally learn these hidden vectors in a bottom-up fashion to learn a classification $\mathcal{W}$, matrix, such that the classification into C sentiment classes becomes the posterior probability of the classes computed as $y^a = softmax(W_s a)$, where $W_s \in \mathbb{R}^{C \times d}$ is the sentiment matrix, multiplied by a word vector a. The word vector a is compused in tri-gram as a parent with two children in a binary tree, and a can be at any level in the tree, the leaves of the hidden computed parents. This allows the word vectors to interact through the non-linearity function. Another model is Matrix-Vector RNN (MV-RNN) which represents words as vectors, and longer phrases as trees, such as each n-gram are represented as a list of (vector, matrix) pairs and a parse tree. This enables the interaction between words in phrases but increases the number of parameters to estimate. RNTN uses a tensor composition function for all nodes such that each slice of the tensor captures a specific type of composition on words' multiplicative interactions. The authors explained the back-propagation algorithm for tensors and used AdaGrad for this non-convex optimisation.

RNTN achieved the highest performance for shorter sentences compared with standard RNN, MV-RNNs, and baselines such as neural networks that use a bag of words ignoring word order and a bag of words features with Naive Bayes and SVMs, as well as Naive Bayes with bag of bigram features. The bag of words method works well with longer sentences. The authors experimented with the different n-grams lengths to show all models' performance. They also experimented with the complexity of sentences such as negation, using "but" to identify the most positive and the most negative sentences. This application uses a neural network model that could have benefitted from more deep layers to capture more complex interactions, reducing hand-crafting of the semantic representation.

As obvious from the previous application, semantic tree approaches are domain-specific and sometimes require huge human experts to define the required relationships manually. The following application falls in between this category (building ternary relationships) and learning an embedding. After reviewing this, we will resume other methods based on learning an embedding from a given text corpus.

The authors in (Weber et al., 2017) presented a natural language understanding application, where tensors are used to capture multiplicative interactions combining predicate, object and subject, generating aggregated representations for event prediction tasks. The RNN models are sequential in events, while additive models create a compound effect of the

variables using additive functions. A generalised additive model can be defined as $y = \beta + f_1(x_1) + f_2(x_2) + \cdots + f_n(x_n) = \sum_{i=1}^{n} f_i(x_i)$, or even $y = \beta + \sum_{i=1}^{n} f_i(x_i) + \sum_{i \neq j} f_{i,j}(x_i, x_j)$ as defined in (Hastie and Tibshirani, 1999). Multiplicative models are more sensitive to small changes in the arguments interactions. Tensor-based provide multiplicative models that were utilised in the Weber et al. paper to learn tensors P of the predicate, subject, and object, to predict an event e computed as tensor contraction P(s, o), such that the event vector e_i, is learned as $e_i = \sum_{ijk} P_{i,j,k} s_j o_k$, from the s and o as subject and object vectors, respectively. The authors used a predefined word embedding to create the tensor $W^{d \times d \times d}$, where d is the dimension of the input embedding, and each row is a word. The predicate tensor P is learned as $P_{ijk} = W_{ijk} \sum_a p_a U_{ajk}$, where $U^{d \times d \times d}$ is a tensor capturing the linear functions for each one-dimensional row of W, determining how the predicate embedding p should scale that dimension. This makes Tensor-based Event Composition (as the first model) predict an event $e_i = \sum_{a,ijk} p_a s_j o_k W_{ijk} U_{ajk}$. The authors used a training data from the New York Times Gigaword Corpus, extracting the event triplets using the Open Information Extraction system Ollie. They initialized the word embedding layer with 100-dimensional pre-trained GloVe vectors. A second model is proposed as the Role Factored Tensor model, such that $e = W_s v_s + W_0 v_s$, such that $v_s = T(s, p)$, and $v_o = T(o, p)$, where $T^{h \times d \times d}$ with h as the size of the output and d as the dimensionality of the embdding, is a tensor composed of factored composition strategy that captures interactions between the predicate and its arguments separately to then combine these interactions into the final embedding. They trained a baseline as a two-layer compositional neural network model $e = W \times \tanh(H[s; p; o])$, where W and H are the model parameters to estimate using Adagrad learning algorithm with a 0.01 learning rate and minibatch size of 128. The second baseline is a multiplicative two-layer NN concatenating the elementwise multiplications between the verb and its subject/object such that $e = W \times \tanh(H[s; p; o; p \odot s; p \odot o])$, where $\odot$ is the ekement-wise multiplication. They created another variant to predict the next word, not just the next event. They evaluated the model using Coherent Multiple Choice Narrative Cloze dataset (CMCNC) which is used to estimate which event has been held out from a document from a small set of randomly drawn events, found at https://paperswithcode.com/dataset/cmcnc. They used another automatic variant to this benchmark (MCNC) and showed that the accuracy of the Role Factor Tensor approach is higher in predicting events and words than the Predicate Tensor and the other two NN baselines using Spearman's correlation. The Predicate Tensor approach was better in one case predicting words (which was always more accurate than predicting events), using Hard similarity scores as a percentage of cases where the similar pair had higher cosine similarity than the dissimilar pair. The authors also used their learned tensor to draw an event schema starting from a seed event and connect all possible events that could occur next using the nearest neighbour algorithm. Their code is published at https://github.com/stonybrooknlp/event-tensors.

The NLP application advances were achieved by using word encoding that captures the multiplicative interactions and considers word order efficiently, and also by using deeper NN layers. NLP advancements grew throughout stages from manually crafting the words' embeddings capturing word semantics relations like GloVe to adding a layer that learns this embedding from a given dataset in RNN or LSTM models, to transfer learning to use pre-trained models and fine-tune to specific applications. The RNN model learns sequential dependence of the input over temporal or spatial dependence, making it useful in word order semantics required in NLP applications. RNNs were used in sequence-to-sequence models such as encoder-decoder NN and for applications such as language translation (input as a sequence of words from the source language and output as a sequence of words at the destination language of variable lengths). The last hidden state of the encoder is the source language embedding, which is based on the decoder to generate the equivalent sentence in the destination language. LSTM uses more complex gates, in which some states are remembered for longer than others, adding another level of sophistication in learning the word order complexities and solving the vanishing gradient problem of RNNs. Then, the attention mechanism evolved so that all encoder layers were exposed to the decoder to prevent the final hidden layer production bottleneck. This requires exposing all states by assigning weights to decide which state to use. These weights are learned from the training dataset. Then comes the Transformer encoder/decoder model. The Transformer encoder has two sublayers, a self-attention layer and a Feed Forward NN layer instead of the RNN (sequential naturally) or LSTM cells (at least four times the computational cost of RNN cells). Transformers use positional encoding to compensate for the missing sequencing due to the elimination of RNNs and LSTMs. Then the decoder component uses the abstract vector representation of the input sequences to generate one word at a time, attending to previously generated words using a similar mechanism to the encoder but adding a third sub-layer to perform multi-head attention over the output of the encoder stack, allowing the model to focus on different positions or sub-spaces.

The self-attention is implemented in many ways to capture interactions between all words in a sentence. The original Google proposal was called the scaled dot-product attention that used three matrices, query Q (representing the current word), key K (representing labels for all the words in the segment to score against to identify relevance to the query) and value V (actual word representations), such that the n token embedding is multiplied by them. Similarity scores S nxn matrix is calculated from the scaled dot-product of Q and K vectors, identifying similar ones with large values. The attention weight W nxn matrix is calculated by normalising the similarity scores S using the softmax function. Then the self-attention layer output is produced by multiplying the weights with the value V vector. Instead of one embedding, three embeddings are created for Q, K, and V independently, and then each one of them is projected through multiple linear projections creating multiple heads for the attention layer. These scores identify the relevance of focusing the attention on what to

remember rather than what to forget in LSTM models. The positional encoding of words in sentences is one of the characteristics of the Transformer model, and it can be absolute positional, relative positional, or rotary positional (Vaswani et al., 2017).

Transformers are considered the most advanced NLP approach with applications in computer vision (Dosovitskiy et al., 2021), automatic speech recognition, time series modelling and other machine learning applications, not NLP applications only. While capturing the input interactions in forward and backward dependence, the Transformer design is also parallelised because of recurrence sequential processing elimination and the use of multi-heads and multilinear mappings. Many Transformer based models have been proposed in the literature; some build on the encoder only, the decoder only, or the encoder/decoder model—some examples include BERT, RoBERTa, GPT-2, and DistilBERT, which combine self-attention and transfer learning. The HuggingFace platform (https://huggingface.co/) provides various architectures using a unified codebase for various ML tasks and datasets. For a tutorial style on the Transformer using Python, read the book (Tunstall et al., 2022).

The Transformers estimate a large number of parameters and can benefit from compression techniques such as parameter sharing across layers and low-rank approximations. These can be achieved by tensor decompositions methods such as Block-Term Tensor Decomposition (BTD), which is proposed by the authors of (Ma et al., 2019). BTD combines both CP decomposition and Tucker decomposition, such that a tensor is decomposed into P Tucker decomposition, each with its core tensor and d factor matrices, such that P is the CP rank. The authors first used Single-block attention based on the Tucker decomposition to use a linear function of a set of vectors. Then they built the multi-head attention using the BTD, enabling parameter sharing across multiple blocks, higher compression (8-times fewer parameters), and lower complexity. They tested using PTB, WikiText-103 and One-billion language modelling tasks, and English-German neural machine translation WMT-2016 to show that their method is more compressed and more accurate than Transformer, Transformer XL, TT-format tensor factorised Transformer model, and other models using RNN, LSTM, and others. Their code is published at https://github.com/szhangtju/The-compression-of-Transformer.

6.3.3 Generative DNN Model

As the previous section shows, an uncompressed deep neural network (DNN) can be reconstructed using the corresponding compressed tensor network representation. Using tensor decomposition approaches, the DNN model can be simplified to achieve the desired trade-off between parameterisation and predictive accuracy. Finally, the compressed tensor network is mapped back into the corresponding compressed DNN. This is not limited to

computer vision or NLP applications but to any NN or DNN architecture solving any particular ML task. Another generative DNN architecture model is the Restricted Boltzmann Machines (RBM) which estimates the probability distribution of various datasets. Mapping an RBM to the Tensor Networks States (TNS) has been successfully applied of (Chen et al., 2018). TNS has been applied in various problems in quantum-many body physics. The physics communities refer to Tensor Chain (TC) decomposition as the Matrix Product State (MPS), which is a special case of the Hierarchical Tucker (HT) decomposition and the simplest TNS. MPS is equivalent to the TT format. The authors applied concepts from the quantum information theory, which has developed at a faster pace in the past decades, to define the necessary and sufficient conditions to transform a TNS into an RBM representing quantum states. For example, they map the RBM weights to terms in tensors to form the TNS, and then represent the TNS as MPS. They further use the RBM undirected probabilistic graphical model structure and employ the conditional independence property to provide an optimal MPS model. They discussed other TNS models, adding more deep layers, and how the number of parameters does not increase while the model performance increases. Their code is published at https://github.com/yzcj105/rbm2mps.

For Python examples applying these concepts, many are contributed as open source in the public domain. For example, Tensorly Python package authors compressed a FC layer using TT format at https://github.com/tensorly/Proceedings_IEEE_companion_notebooks/blob/master/tt-compression.ipynb.

They also have a Tensor Regression Layer (TRL) https://github.com/tensorly/Proceedings_IEEE_companion_notebooks/blob/master/tensor_regression_layer.ipynb

A Tensorial RNN can be found at https://github.com/Tuyki/TT_RNN that includes FC, Simple RNN, LSTM and GRU in their Tutorials using PyTorch. Many python packages implement the TT decomposition, such as scikit_tt (https://github.com/PGelss/scikit_tt/)

The Fully Connected layer tensorization and the CNN layer tensorization are implemented in Python and published at https://github.com/timgaripov/TensorNet-TF/tree/master/experiments/cifar-10/FC-Tensorizing-Neural-Networks, and https://github.com/timgaripov/TensorNet-TF/tree/master/experiments/cifar-10/conv-Ultimate-Tensorization.

6.3.4 Multi-modal Neural Networks & Data Fusion Techniques

Multi-modal problems rely on two or more datasets, each coming from its domain and representation requirements. A data fusion step is required to create a unimodal projection out of the multi-modal different spaces representation, capturing the multi-way interactions between all modalities. A simple approach is concatenating the vectors or applying an element-wise sum or product between the different modalities. This will not capture complex interactions between the different modalities. Outer-Product methods are used to capture bilinear interactions between all elements of two vectors, such as an outer product $q \otimes v$ between visual v and textual q embeddings. This approach will generate a massive number of parameters to learn. For example, for two modalities with dimensions $n_1 = n_2 =$ 2048 and the dimension of the weights matrix linearised is $z = 3000$, the number of parameters is 12.5 billion. Multi-modal Compact Bilinear pooling (MCB) uses FFT to further compress the outer product (Fukui et al., 2016). For more than two modalities, more compression, expressive, and interpretable fusion, the tensor multi-way analysis is an intuitive solution to these applications.

The authors in (Ben-younes et al., 2017) address the Visual Question Answering (VQA) task by using tensors to fuse visual and textual representations. They proposed a multi-modal tensor-based Tucker decomposition to capture the interactions between images and textual modalities with fewer parameters (compression) than other bilinear models. The images' internal representation v is learned using a CNN architecture; the textual representation q is learned using GRU sequential architecture. Then a Tucker representation $\mathcal{T} = \left[T_c; W_q, W_v, W_o\right]$ with a core tensor T_c, text matrix W_q, image matrix W_v and the output matrix W_o. The output vector $y = \left(\left(T_c \times_1 \left(q^T W_q\right)\right) \times_2 \left(v^T W_v\right)\right) \times_3 W_o$ produces an answer. They tested the model using the VQA dataset that can be downloaded from https://visualqa.org/. They compared the performance with other state-of-the-art models to show performance improvements. Their code is published at https://github.com/Cadene/vqa.pytorch.

For thre modalities examples, the work in (Li et al., 2020) created a multi-modal sentiment analysis (MSA) using the MOSI/CMU-MOSI dataset of a of the form (A, V , L), where A = {A_1, . . . , A_T }, V = {V_1, . . . , V_T } and L = {L_1, . . . ,L_T }, denote the time series of the length T w.r.t. the acoustic, visual and language data, respectively. The dataset is published at https://paperswithcode.com/dataset/multimodal-opinionlevel-sentiment-intensity and https://github.com/A2Zadeh/CMU-MultimodalSDK. The aim is to learn the composite function $\hat{y} = f(\varphi_a(A), \varphi_v(V), \varphi_l(L))$where $\varphi_i(X)$ is the sub-mapping from the raw data to the features. This function includes the fusion phase and is learned by an LSTM model using a preprocessed features representation that keeps the uni-modal representation and concatenated data from the different modalities for each time step, then across k-time steps. They proposed Time Product Fusion Network (TPFN) that builds on the temporal

tensor fusion network (T2FN). TPFN applies implicit outer product methods across sliding time windows to capture the model interaction across modalities in the data fusion phase. CP is the method for low-rank decomposition, and regularisation on the low-rank representation handles incomplete datasets. Their code is published at https://qibinzhao.github.io/publications/ECCV2020_LiChao/TPFN.zip.

In (Hou et al., 2019), the authors addressed the MSA problem by proposing a High-order polynomial tensor pooling (PTP). PTP concatenated features form a Tensor by tensor product operation of order P to represent all possible polynomial expansions up to order P. As P increases, so does the number of parameters to learn, but the higher polynomial interactions between tensors that can be captured. Using CP decomposition, the weights tensor is compressed. Then Hierarchical polynomial fusion network (HPFN) is formed assuming 2D feature map time series. HPFN recursively learn the local temporal modalities pattern by arranging PTP in multiple layers. This borrows many features from CNN, including receptive fields, sharing parameters, scanning window, and PTP 'fusion filters'. Their code is published at https://qibinzhao.github.io/publications/NeurIPS_2019_HouMing/HPFN.zip.

Tensorising Activation functions:

In the first building block of a neural network design, the choice of activation function is typically achieved by a weighted sum of the inputs for vectorial data using the inner product between the weight vector and the input vector. In tensorizing the neural network, the tensors can be used to modify input aggregation functions to be suitable for input in tree-structured data, such that a specific node in the tree, the neuron, recursively computes its activation by a weighted sum of the activations of its children, with appropriate weight sharing assumptions. Such an aggregation function can be easily tensorized. Working with tree data structures usually requires recursion and graph data structures, which will be discussed in the following section.

6.3.5 GNN applications

This chapter reviewed the advances in Deep Neural Networks (DNN) and how tensor decompositions have been applied to them. Graph Neural Networks (GNN) and how graph data structures have been applied in DNN and using Tensor decomposition approaches using graph and network data structures have been applied in the literature. NN build the computational graph as a multipartite graph; however, using graph input data structures to train a deep neural network enables various graph theory and network analytics algorithms to benefit from the depth and non-linearity of the network.

The authors of (Kwon and Chung, 2022) proposed a recursive tensor decomposition method that is based on the CP decomposition by choosing orthogonal vectors in the SVD step creating a decomposition tree. They experimented on MNIST, CIFAR-10, and ILSVRC

2012 datasets, to achieve a 154× reduction in weight parameters with only a 1% accuracy drop compared to the original baselines for these datasets. This method is more suitable for the neuromorphic systems that will be reviewed in the next chapter.

The work in (Hamdi and Angryk, 2019) presents tensor decomposition-based node embedding algorithms that learn node features from arbitrary types of graphs: undirected, directed, and/or weighted, without relying on computationally expensive eigendecomposition or requiring tuning of the word embedding-based hyperparameters as a result of representing the graph as a node sequence similar to the sentences in a document.

The work in (Jermyn, 2017) presents tensor trees as efficient tensor computer representations based on both optimal brute force and greedy algorithm heuristic that performs well for higher rank tensors tree decompositions.

Based on these advances, I find the advances in Graph Neural Networks (GNN) to be complementing tensor decomposition and their applications in DNN. GNN is another active research topic and almost reaching maturity, as presented in (Liu and Zhou, 2020). The book explains how various architectures of DNN (CNN, LSTM, Attention, Residual and Hetregenous) are constructed from graph and network analysis algorithms. The GNN builds NN from graph structures with node and edge attributes and can use different representations such as real-valued vectors and tensors. The introduction to GNN presented in (Bacciu et al., 2020) identifies Graph Neural networks as recursive neural units with cycles between the node states to capture the mutual dependence, while the term Neural Network for Graphs defines the architecture that captures the mutual dependence through layers and passing on representations while eliminating recursion. The authors provide a tutorial on Deep Graph Networks (DGN) and variants such as Bayesian and Generative.

The advantages of GNN are closely related to the compressive DNN tensor decomposition methods, such as using traditional spectral graph theory to reduce the computational cost of shared weights on one side. On the other side, the hierarchical patterns that capture features of different sizes can be represented with multi-layer graph structures. For graph transformation, graph-tensors proposed in (Malik et al., 2019) learn embeddings of time-varying graphs based on a tensor framework. There are also matrix networks proposed in (Sun et al., 2018) and graph tensor neural networks (Liu and Zhu, 2021). Spektral is a framework of GNN different models that are developed at https://graphneural.network/. The Deep Graph Library (DGL) is a framework for different GNN models that scale to large graphs using GPUs and distributed architectures. Their codebase and examples are published at https://www.dgl.ai/. PyTorch geometric is another GNN framework hosted at https://www.pyg.org/. Nvidia offer platforms to parallelise the training of DGLs and PyG

GNNs on GPUs using Memgraph, cuGraph, and graph-as-a-service, as illustrated in https://developer.nvidia.com/gnn-frameworks.

6.4 General Framework

The field is far from established, and a general framework can be precisely defined at this stage. However, general steps to organise the process of building a multi-way analysis model or compressing an existing model can be proposed, subject to many possibilities at every step, and open for creativity and research outcome proposals. Below is an attempt to define sample steps to tensorize ML and DL tasks:

- Make sure Data is reformatted in Tensor format or can be formatted.
    - Choose already formatted multi-way data, such as
        - http://www.models.life.ku.dk/nwaydata,
        - https://three-mode.leidenuniv.nl/.
        - https://github.com/zhaoxile/reproducible-tensor-completion-state-of-the-art.
    - Create a script to collect the data in the required format. fMRI, EEG and similar data, signal processing data, and multiset data in which at least one of the modes consists of different entities in each level are all types of data that can be used to create a tensor format.
    - Extract data using a network/graph dataset that is already multi-way.
        - For NLP data, ontologies, RDFs or platforms such as https://github.com/knowitall/ollie can be used.
    - Apply a tensorisation step such as Binarisation, Segmentation, decimation, folding, reshaping, Hankelization, multi-way Toeplitz Löwner and higher-order statistics (Debals, 2017).
        - Sometimes the data contains enough information to connect all modes of tensorisation; for example, the time mode is naturally related to the spatial modes in videos. Other datasets might require adding an extra variable to connect the modes, for example, adding a time mode of an experiment to tensorise a set of experiments each in matrix form, or an extra variable such as illumination/poses in tensor faces to connect the Eigenfaces matrices.
    - Make sure the created tensor properties are suitable to the original data properties, tensor decomposition readiness and to the required analysis to be done.

- Standardise all data to centre the mean around zero and rescale to a standard deviation of one. Some tensorisation steps will benefit from standardisation on the original dataset.
- Handling large datasets whose tensors can not be created in memory:
    - Choose random samples to create the tensor from, or block sampling.
    - Use incomplete tensors.
    - Apply some dimensionality reduction method on the original dataset as explained in chapter two or representation learning method as explained in chapter five to transform the data into a compact form better than randomization.
    - Attempt identifying the components of the tensor decomposition that you can multiply together to resume with a reconstructed tensor from those components, such as the method implemented in the Tensorfaces paper. This is called implicit tensorization, which combines tensorization with tensor decompositions without the explicit construction of a tensor. Tensor recognition as well is the process of identifying an implicit tensor and the ability to construct it from a given dataset (tensor representing multi-way dataset) or a problem definition (tensor representing multi-linear function and polynomials). Tensor recognition is a skill that can be gained by exposure to various tensorised problems and approaches (Debals, 2017).
    - Tensor networks can be diagrammatically drawn, and the tensor contraction code is generated, as shown in the tool published at https://www.tensortrace.com/.
- Turn into the tensor form using examples from the Python code in "tensorisation.ipynb" and "multi-wayExamples.ipynb".
- To achieve dimensionality reduction, use a tensor decomposition approach.
- Choose the machine learning model to apply. Pass only the core decomposed tensors for the data and compute the metrics to evaluate.
- For DNN models, choose where to apply the tensorization steps discussed earlier and experiment with the performance evaluation to identify the most suitable for a given problem and dataset.
- Using transparent, simpler code built from scratch for NN and DNN enables complete control over all computation and to have code that can survive the many waves of the existing DNN platforms' evolution and broken compatibility. For example, the following are simpler models: https://github.com/Sentdex/NNfSiX , https://github.com/ahmedfgad/NumPyANN, https://github.com/ahmedfgad/NumPyCNN, https://github.com/revsic/numpy-rnn, https://github.com/pangolulu/rnn-from-scratch, https://github.com/CaptainE/RNN-LSTM-in-numpy, https://github.com/3outeille/GANumpy, and many more.

Otherwise, keep a virtual machine with all dependencies and do not update any module independently.

References

Acar, E., Çamtepe, S.A., Krishnamoorthy, M.S., Yener, B., 2005. Modeling and Multiway Analysis of Chatroom Tensors, in: Kantor, P., Muresan, G., Roberts, F., Zeng, D.D., Wang, F.-Y., Chen, H., Merkle, R.C. (Eds.), Intelligence and Security Informatics, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 256–268. https://doi.org/10.1007/11427995_21

Acar, E., Çamtepe, S.A., Yener, B., 2006. Collective Sampling and Analysis of High Order Tensors for Chatroom Communications, in: Mehrotra, S., Zeng, D.D., Chen, H., Thuraisingham, B., Wang, F.-Y. (Eds.), Intelligence and Security Informatics, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 213–224. https://doi.org/10.1007/11760146_19

Armstrong, M.D.S., Hinrich, J.L., de la Mata, A.P., Harynuk, J.J., 2022. PARAFAC2×N: Coupled Decomposition of Multi-modal Data with Drift in N Modes.

Bacciu, D., Errica, F., Micheli, A., Podda, M., 2020. A Gentle Introduction to Deep Learning for Graphs. https://doi.org/10.1016/j.neunet.2020.06.006

Bader, B.W., Berry, M.W., Browne, M., 2008. Discussion Tracking in Enron Email Using PARAFAC, in: Berry, M.W., Castellanos, M. (Eds.), Survey of Text Mining II. Springer London, London, pp. 147–163. https://doi.org/10.1007/978-1-84800-046-9_8

Ben-younes, H., Cadene, R., Cord, M., Thome, N., 2017. MUTAN: Multimodal Tucker Fusion for Visual Question Answering. Proceedings of the IEEE international conference on computer vision 2612–2620.

Calvi, G.G., Moniri, A., Mahfouz, M., Zhao, Q., Mandic, D.P., 2020. Compression and Interpretability of Deep Neural Networks via Tucker Tensor Layer: From First Principles to Tensor Valued Back-Propagation. arXiv:1903.06133 [cs, eess].

Castellana, D., Bacciu, D., 2019. Bayesian Tensor Factorisation for Bottom-up Hidden Tree Markov Models.

Chen, J., Cheng, S., Xie, H., Wang, L., Xiang, T., 2018. Equivalence of restricted Boltzmann machines and tensor network states. Phys. Rev. B 97, 085104. https://doi.org/10.1103/PhysRevB.97.085104

Debals, O., 2017. Tensorization and Applications in Blind Source Separation.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., de Freitas, N., 2014. Predicting Parameters in Deep Learning.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N., 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.

Fan, A., Grave, E., Joulin, A., 2019. Reducing Transformer Depth on Demand with Structured Dropout.

Fukui, A., Park, D.H., Yang, D., Rohrbach, A., Darrell, T., Rohrbach, M., 2016. Multimodal Compact Bilinear Pooling for Visual Question Answering and Visual Grounding.

Gabor, M., Zdunek, R., 2022. Compressing Convolutional Neural Networks with Hierarchical Tucker-2 Decomposition. SSRN Journal. https://doi.org/10.2139/ssrn.4031519

Gong, Y., Liu, L., Yang, M., Bourdev, L., 2014. Compressing Deep Convolutional Networks using Vector Quantization.

Hamdi, S.M., Angryk, R., 2019. Interpretable Feature Learning of Graphs using Tensor Decomposition, in: 2019 IEEE International Conference on Data Mining (ICDM). Presented at the 2019 IEEE International Conference on Data Mining (ICDM), IEEE, Beijing, China, pp. 270–279. https://doi.org/10.1109/ICDM.2019.00037

Hastie, T., Tibshirani, R., 1999. Generalized additive models. Chapman & Hall/CRC, Boca Raton, Fla.

Hou, M., Tang, J., Zhang, J., Kong, W., Zhao, Q., 2019. Deep Multimodal Multilinear Fusion with High-order Polynomial Pooling, in: Advances in Neural Information Processing Systems. Curran Associates, Inc.

Jermyn, A.S., 2017. Efficient Decomposition of High-Rank Tensors. https://doi.org/10.1016/j.jcp.2018.10.026

Knodt, J., 2022. Structural Dropout for Model Width Compression.

Kroonenberg, P.M., 2008. Applied multiway data analysis, Wiley series in probability and statistics. Wiley-Interscience, Hoboken, N.J.

Kwon, K., Chung, J., 2022. Reducing Parameters of Neural Networks via Recursive Tensor Approximation. Electronics 11, 214. https://doi.org/10.3390/electronics11020214

Lacroix, T., Obozinski, G., Usunier, N., 2020. Tensor Decompositions for temporal knowledge base completion. Presented at the The International Conference on Learning Representations (ICLR).

Lantsov, V.N., 2021. Solving the circuit equations using tensor decompositions. J. Phys.: Conf. Ser. 1889, 022097. https://doi.org/10.1088/1742-6596/1889/2/022097

Lehky, S.R., Phan, A.H., Cichocki, A., Tanaka, K., 2020. Face Representations via Tensorfaces of Various Complexities. Neural Computation 32, 281–329. https://doi.org/10.1162/neco_a_01258

Li, B., Li, C., Duan, F., Zheng, N., Zhao, Q., 2020. TPFN: Applying Outer Product along Time to Multimodal Sentiment Analysis Fusion on Incomplete Data. Presented at the 16th european conference on. COMPUTER VISION, p. 17.

Liu, X.-Y., Zhu, M., 2021. Convolutional Graph-Tensor Net for Graph Data Completion. Presented at the IJCAI 2020 Workshop on Tensor Network Represenations in Machine Learning, arXiv.

Liu, Z., Zhou, J., 2020. Introduction to Graph Neural Networks. Synthesis Lectures on Artificial Intelligence and Machine Learning 14, 1–127. https://doi.org/10.2200/S00980ED1V01Y202001AIM045

Ma, X., Zhang, P., Zhang, S., Duan, N., Hou, Y., Song, D., Zhou, M., 2019. A Tensorized Transformer for Language Modeling, in: Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Presented at the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), arXiv, Vancouver, Canada.

Malik, O.A., Ubaru, S., Horesh, L., Kilmer, M.E., Avron, H., 2019. Tensor Graph Neural Networks for Learning on Time Varying Graphs. Presented at the NeurIPS Workshop on Graph Representation Learning, p. 7.

Nair, V., Chatterjee, M., Tavakoli, N., Namin, A.S., Snoeyink, C., 2020. Optimizing CNN using Fast Fourier Transformation for Object Recognition. Presented at the e IEEE International Conference on Machine Learning Applications (ICMLA'20), p. 10.

Nickel, M., Tresp, V., Kriegel, H.-P., 2012. Factorizing YAGO: scalable machine learning for linked data, in: Proceedings of the 21st International Conference on World Wide Web. Presented at the WWW 2012: 21st World Wide Web Conference 2012, ACM, Lyon France, pp. 271–280. https://doi.org/10.1145/2187836.2187874

Nickel, M., Tresp, V., Kriegel, H.-P., 2011. A three-way model for collective learning on multi-relational data, in: Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML'11. Omnipress, Madison, WI, USA, pp. 809–816.

Novikov, A., Podoprikhin, D., Osokin, A., Vetrov, D., 2015. Tensorizing Neural Networks. NIPS-2015, Advances in Neural Information Processing Systems 28.

Padia, A., Kalpakis, K., Finin, T., 2016. Inferring relations in knowledge graphs with tensor decompositions, in: 2016 IEEE International Conference on Big Data (Big Data). Presented at the 2016 IEEE International Conference on Big Data (Big Data), IEEE, Washington DC,USA, pp. 4020–4022. https://doi.org/10.1109/BigData.2016.7841096

Pan, Y., Xu, J., Wang, M., Ye, J., Wang, F., Bai, K., Xu, Z., 2018. Compressing Recurrent Neural Networks with Tensor Ring for Action Recognition.

Schlüter, F., 2014. A survey on independence-based Markov networks learning. Artif Intell Rev 42, 1069–1093. https://doi.org/10.1007/s10462-012-9346-y

Smilde, A., Bro, R., Geladi, P., 2004. Multi-Way Analysis with Applications in the Chemical Sciences. John Wiley & Sons, Ltd, Chichester, UK. https://doi.org/10.1002/0470012110

Socher, R., Perelygin, A., Wu, J.Y., Chuang, J., Manning, C.D., Ng, A.Y., Potts, C., 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank, in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. Seattle, Washington, USA, pp. 1631–1642.

Sun, Q., Yan, M., Donoho, D., boyd, 2018. Convolutional Imputation of Matrix Networks, in: Proceedings of the 35th International Conference on Machine Learning. Presented at the International Conference on Machine Learning, PMLR, pp. 4818–4827.

Sutskever, I., Salakhutdinov, R., Tenenbaum, J.B., n.d. Modelling Relational Data using Bayesian Clustered Tensor Factorization 8.

Tunstall, L., Werra, L. von, Wolf, T., Géron, A., 2022. Natural language processing with Transformers: building language applications with Hugging Face. O'Reilly, Cambridge.

Turk, M.A., Pentland, A.P., 1991. Face recognition using eigenfaces, in: Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Presented at the 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE Comput. Sco. Press, Maui, HI, USA, pp. 586–591. https://doi.org/10.1109/CVPR.1991.139758

Vasilescu, M.A.O., 2002. Human motion signatures: analysis, synthesis, recognition, in: Object Recognition Supported by User Interaction for Service Robots. Presented at the 16th International Conference on Pattern Recognition, IEEE Comput. Soc,

Quebec City, Que., Canada, pp. 456–460. https://doi.org/10.1109/ICPR.2002.1047975

Vasilescu, M.A.O., Terzopoulos, D., 2003. Multilinear subspace analysis of image ensembles, in: 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings. Presented at the CVPR 2003: Computer Vision and Pattern Recognition Conference, IEEE Comput. Soc, Madison, WI, USA, p. II-93–9. https://doi.org/10.1109/CVPR.2003.1211457

Vasilescu, M. Alex O., Terzopoulos, D., 2002. Multilinear Analysis of Image Ensembles: TensorFaces, in: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (Eds.), Computer Vision — ECCV 2002, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 447–460. https://doi.org/10.1007/3-540-47969-4_30

Vasilescu, M.A.O., Terzopoulos, D., 2002. Multilinear image analysis for facial recognition, in: Object Recognition Supported by User Interaction for Service Robots. Presented at the 16th International Conference on Pattern Recognition, IEEE Comput. Soc, Quebec City, Que., Canada, pp. 511–514. https://doi.org/10.1109/ICPR.2002.1048350

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2017. Attention Is All You Need.

Wang, J., Wang, S., Wang, Y., Hu, H., Yu, J., Zhao, X., Liu, J., Chen, X., Li, Y., 2021. A data process of human knee joint kinematics obtained by motion-capture measurement. BMC Medical Informatics and Decision Making 21, 121. https://doi.org/10.1186/s12911-021-01483-0

Weber, N., Balasubramanian, N., Chambers, N., 2017. Event Representations with Tensor-based Compositions.

Wu, B., Wang, D., Zhao, G., Deng, L., Li, G., 2020. Hybrid Tensor Decomposition in Neural Network Compression. arXiv:2006.15938 [cs]. https://doi.org/10.1016/j.neunet.2020.09.006

Yang, Y., Krompass, D., Tresp, V., 2017. Tensor-Train Recurrent Neural Networks for Video Classification.