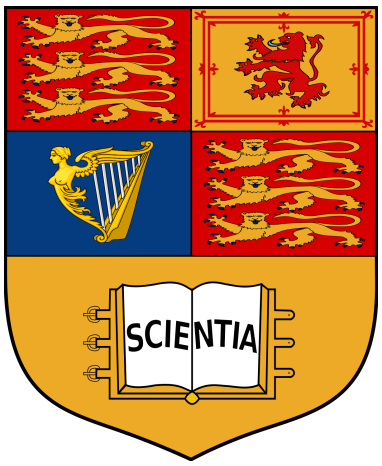
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2021



Project Title:	**A Big Data Approach to Financial Forecasting Through Tensors**
Student:	**Niklas S. Kuusisto**
CID:	**01358208**
Course:	**EIE4**
Project Supervisor:	**Prof D.P. Mandic**
Second Marker:	**Dr C. Ling**

Final Report Plagiarism Statement

I affirm that I have submitted, or will submit, an electronic copy of my final year project report to the provided EEE link.

I affirm that I have provided explicit references for all the material in my Final Report that is not authored by me, but is represented as my own work.

Acknowledgements

I would like to thank my supervisor Professor Danilo Mandic for guiding and providing feedback for me.

I would also like to thank my second marker Dr Cong Ling who helped me with understanding the project requirements and getting started with the project.

Finally, I would like to thank my family for supporting and helping me throughout my studies and our dog, Felix, who recently passed away. Felix brought joy into my life and his memory will live on forever.

Abstract

This project studies how effective different tensorization methods are in forecasting financial time series of high-dimensions. After presenting and testing various tensorization methods, the scope is limited to one tensorization method, namely, Hankel folding or Hankelization. Machine learning (ML) approach is used in the forecasting through the use of recurrent neural networks (RNNs). Furthermore, tensor decomposition (TD) through tensor-train (TT) decomposition is used which helps with compressing the data and making it more interpretable, among other things. Thus, the precise research question is how effective is Hankelization combined with TT decomposition and RNN in financial time series forecasting. To answer this research question, an in-depth background relating to tensors and their operations as well as forecasting methods are presented after which examples of the various tensorization methods are shown. Then an empirical study is conducted in which returns of S&P 500 index (SPX) are predicted using different RNN models. The models are evaluated according to model complexity, loss and accuracy as well as chosen trading metrics. The simulation results show that by Hankelizing the input tensor and forecasting using a tensor-train recurrent neural network model, the forecasting accuracy is further improved. Moreover, by using TT decomposition, this offers compression and regularization properties for the model as well as mitigates the black-box nature of neural networks (NNs) by making the results more interpretable.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

This project explores a big data approach to forecasting using tensors with a financial application. Forecasting is a vital tool as it helps people and businesses in their decision making as well as developing strategies. A typical forecasting includes training models based on historical data. These models are then used to predict future trends. For example, forecasting allows businesses to predict their sales turnover and thus profits as well as employment needs and changes in inventories. On a macroeconomic level, forecasts are used to predict future conditions of the economy such as gross domestic product, net exports, inflation and unemployment. These forecasts are used to determine fiscal and monetary policies such as adjusting interest rates. Financial institutions also rely on forecasts when they try to predict share prices of stocks and values of indices and develop sector rotation strategies, among other things. Accurate forecasts can thus help financial institutions and investors in their portfolio maximization problems and lead to greater profits. [9]

Big data refers to the collection of datasets that consist of massive unstructured, semi-structured and structured data. Big data is often characterised through four Vs of volume, variety, veracity and velocity. These four Vs are the challenges facing big data and they stand for amount of data, range of data types and sources, data quality and speed of incoming data, respectively.

[10] Today, outstanding amounts of data are being produced and the traditional data-analysis techniques are not sufficient. Solving today's problems require more than the traditional techniques can offer.

To solve the challenges of volume, variety, veracity and velocity facing Big Data, tensors can be used. Tensors are multi-dimensional arrays consisting of different modes such as time, frequency, trials and classes [11]. Moreover, TD methods are valuable as they provide ways to compress data, thus allowing for extraction of data from tensors and making the data more interpretable. Furthermore, TD methods allow one to solve many high-dimensional data and machine learning problems without suffering from the curse of dimensionality [1]. For example, TD techniques may be used to extract complex relationships from prices' time series [12]. Financial data is of high dimensions naturally and comes in large volume and fast velocity and are thus a good application for the use of tensors.

1.2 Research Question

Previously not much emphasis has been on different tensorization methods applied in a financial context. Motivated by this, Hankelization combined with TT format and using a ML approach through RNNs in financial forecasting is explored in this project. As such, this project tries to answer how effective such an approach is in financial forecasting. The research question is answered by forecasting the next day returns of the SPX.

1.3 Project Scope

Forecasting methods implemented in empirical part of this project include variations of a ML model, namely, RNN. Autoregressive integrated moving average (ARIMA) and generalized autoregressive conditional heteroskedasticity (GARCH) models are also outlined in chapter 2 but their applications combined with different tensorization and TD methods are beyond the scope of this project. However, they offer possible directions for future work. Several

tensorization methods presented in [13] are considered in this project but ultimately the scope is limited to Hankelization in the empirical part. This is because Hankelization showed promise in [8] when forecasting multiple short time series. Similarly, there exist many TD methods but the scope is limited to only TT decomposition method in the empirical part. However, Tucker decomposition method is explained in chapter 2 and it also provides a possible direction for future work by replacing TT decomposition with Tucker decomposition.

1.4 Report Structure

This report is structured as follows. Chapter 2 explains the theoretical background consisting of tensors, forecasting methods and financial terminology. Chapter 3 illustrates the tensorization methods with some examples. Chapter 4 presents the data used in the empirical part and explains any pre-processing done to the data. Chapter 5 reviews implementations of the NN models. Chapter 6 presents, analyses and compares the results. Finally, chapter 7 concludes this report and provides possible directions for future work.

1.5 Related Literature

Financial time series forecasting is a large field for academia and financial industries. This is due to its several implementation areas as well as because it provides a significant economic impact if the price movements can be predicted more accurately. Therefore many attempts have been made at trying to produce increasingly better financial time series forecasts. Before the research focus shifted more towards ML and deep learning (DL) methods as well as tensor based approaches, support vector machines (SVM) provided and still provide a viable forecasting alternative.

[14] applied SVM to predict the stock price index. The reason to use SVMs was that they use a risk function that consists of the empirical error and a regularized term being derived from the structural risk minimization principle. In [14] the performance of the SVM was

compared with back-propagation neural network (BPN) and case-based reasoning (CBR). The empirical results showed that the SVM outperformed both the BPN and CBR and it was therefore concluded that SVMs provide a viable alternative for financial time series forecasting. Moreover, [15] also applied SVM to predict the stock price index. The performance of the SVM was compared with artificial neural network (ANN) and ARIMA methods. They also concluded that the SVM outperformed both methods by providing a more accurate model and thus a more efficient forecasting method.

To take SVMs further, [16] implemented a support tensor machine (STM) for financial forecasting. Financial indices are multivariate by nature which means that vector-based SVMs lead to a loss of information. This is because SVMs are unable to utilise the available multi-way data structure of financial indices. A tensor extension of SVM was thus implemented in [16] as tensors have superior structural information content over vectors and it was applied in a financial context. In doing so, a least-squares formulation of STM (LS-STM) was implemented in which tensorized data data included the implied index of volatility of the SPX, GC1 gold commodity and SPX. The results indicated that the LS-STM implementation outperformed a standard SVM according to accuracy rate, annualized Sharpe ratio and annualized volatility.

Recently ML and DL approaches such as RNN and long short-term memory (LSTM) in financial time series forecasting have gained popularity partly due to advances in GPU computing. The state-of-the-art approaches in financial time series forecasting include DeepAR (DL based approach) and XGBoost (ML based approach). In [17] DeepAR was implemented which is a methodology for producing accurate probabilistic forecasts based on training an autoregressive RNN model on a large number of related time series. [17] showed that DeepAR produces more accurate forecasts than other state-of-the-art methods when tested on several real-world forecasting datasets. Extreme gradient boost, or XGBoost, on the other hand is an ensemble tree method and was introduced for better speed and performance [18]. In [18] it was demonstrated that XGBoost had a good performance in predicting stock market prices according to error measures (including root-mean-square error) and often outperformed other tree-based models such as gradient boosting, random forest and decision tree.

ML and DL methods are often criticized due to their black-box nature. Therefore, the results they produce can be difficult to interpret and judge. Moreover, to train good forecasting models, ML and DL based approaches require vast amount of data which can be a burden particularly in the training phase when it comes to time and memory required. Tensor based approaches in financial time series forecasting offer viable solutions to the limitations of black-box models as they make the data more interpretable through TD and can handle high-dimensional data. TD allows one to extract complex relationships from stock prices' time series without the need to aggregate the data [12]. Using TD can therefore avoid losses of information as the original time-varying nature of the records is retained [12]. Furthermore, in [12] it was concluded that using TD method in portfolio maximization problem induces investment plans that are superior to simpler investment strategies.

There have recently been efforts to incorporate tensors into ML and DL based methods with various objectives. To reduce the number of parameters while preserving the expressive power of the of fully-connected layers in neural networks (NN), in [19] tensorized NNs by using TT format as a compression tool was implemented. [3] then applied TT decomposition method for video classification and achieved competitive performances with state-of-the-art models with a less complex model architecture. [20] used low-rank matrix factorization in deep neural networks (DNN) and was able to reduce the number of parameters of the network by 30-50% without suffering a significant loss in the accuracy. [21] came to the same conclusion that using low-rank tensor approximations in tensor regression networks achieve high compression rate while only having a slight impact on the accuracy. Moreover, [22] instead used CANDECOMP/PARAFAC (CP) decomposition to speed up convolutional neural networks (CNN). The results showed that by using CP decomposition in CNN, a 8.5× CPU speedup of the whole network with a trade-off of a 1% decrease in the accuracy was obtained. Additionally, to mitigate inherent black-box nature of ML and DL based methods, [23] combined TT decomposition with RNN. Such a method was able to compress the NN and offer regularization properties while achieving better accuracy and making the model more interpretable when compared with its uncompressed RNN counterpart and other tensor based forecasting methods [23].

Instead of employing ML or DL approach, [8] developed a block Hankel tensor ARIMA method.

This exploited the low-rank structure of block Hankel tensors and efficiently captured intrinsic correlations among multiple time series. Such a method can improve the forecasting results, particularly in the case of multiple short time series. [8] experimented the method on several different datasets and concluded that the block Hankel tensor ARIMA method led to superior forecasting performance and reduced computational cost when compared with current state-of-the-art methods of DeepAR and XGBoost.

There has been research conducted on different tensorization methods but little emphasis has been on their application in a financial context, for example, in forecasting stock prices or volatility of stocks. [13] laid out different tensorization methods such as reshaping or folding, tensorization through a Toeplitz/Hankel tensor, tensorization by means of Löwner matrix (Löwner folding), tensorization based on cumulant and derivatives of the generalized characteristic functions and tensorization by learning local structures, among other methods. For a more exhaustive review on the different tensorization methods, please refer to [13]. In this project some of the tensorization methods presented in [13] are reviewed and Hankelization tensorization technique is combined with NNs and TT decomposition method to forecast in a financial context.

Chapter 2

Background

In this chapter the necessary background that is used throughout the report is outlined. In doing so, first tensor notations, operations, tensorization methods and tensor decomposition are discussed. After that the forecasting methods are presented and useful financial vocabulary explained. Having read this chapter, the reader will have all the necessary background information to follow the rest of this report.

2.1 Tensors

Tensors are of great importance to this project and having an understanding of them is fundamental before proceeding further. This section gives an overview on some of the tensor notations and operations. For a more exhaustive review on tensor notations, conventions and operations, please refer to [24] and [2].

2.1.1 Notations

Tensors are multi-dimensional generalisation of matrices. The order of a tensor is the number of its dimensions or modes. These modes could include, for example, time, frequency, trials

and classes. More familiar cases of tensors include scalars (zero-order tensors), vectors (first-order tensors) and matrices (second-order tensors). For example, the two modes of a matrix are rows and columns. A third order tensor has three modes and resembles a cube. Following this principle, an Nth order tensor has N modes. Figure 2.1 shows graphical representations of tensors with orders ranging from 0 to 4. In the figure it can be seen that the fourth-order tensor is a vector of third-order tensors.

A real-valued tensor with N modes is mathematically denoted as $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$. A specific entry of the tensor is denoted as $a_{i_1,i_2,\ldots,i_N} = \mathcal{A}(i_1, i_2, \ldots, i_N) \in \mathbb{R}$. When discussing about the size of a tensor, this refers to the number of values that an index can take in a particular mode. Tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ is of order N and size I_n in all modes-n where $n = 1, 2, \ldots, N$.

Taking a subset of the original data tensor, subtensors are formed. More specifically, when a fixed subset of indices are used, subtensors are formed. There exists a specific type of subtensors called *fibers*. These are vector-valued subtensors that are defined by fixing every index but one. Similarly, by fixing two indices, matrix-valued subtensors are obtained, called *slices*. Figure 2.2 shows graphical representations of a tensor, slice and fiber. Table 2.1 summarizes the basic tensor notations discussed above.

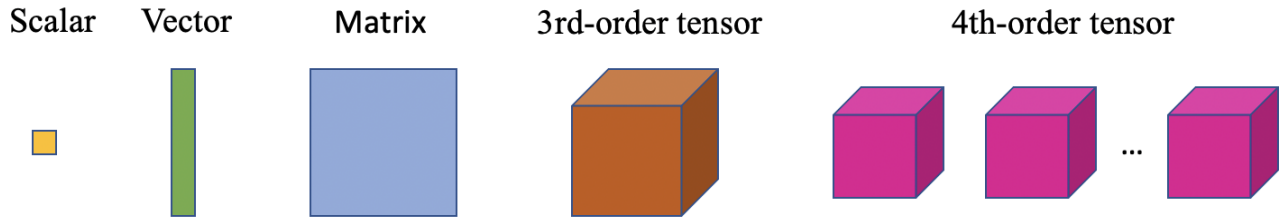


Figure 2.1: Graphical representations of tensors of order 0, 1, 2, 3 and 4.

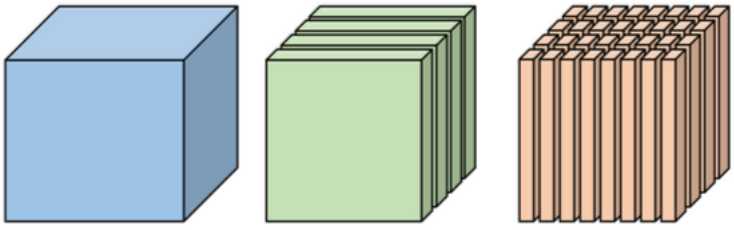


Figure 2.2: Graphical representations of a tensor, a (frontal) slice and a (column) fiber. Source: adapted from [1].

Table 2.1: Tensor notations summarized

a, $\mathbf{a}$, $\mathbf{A}$, $\mathcal{A}$	scalar, vector, matrix and tensor
$\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times ... \times I_N}$	tensor of order N of size $I_1 \times I_2 \times ... \times I_N$
$\mathbf{a}(:, i_2, i_3, ..., i_N)$	fiber of tensor $\mathcal{A}$
$\mathbf{A}(:, :, i_3, ..., i_N)$	matrix slice of tensor $\mathcal{A}$
$\mathcal{A}(:, :, :, i_4, ..., i_N)$	subtensor of $\mathcal{A}$ that has several indices fixed

2.1.2 Operations

Reshaping or reformatting is the simplest way of tensorization and it is a unique tensor operation which changes the tensor's shape but not the underlying data of the tensor. Tensor reshaping can also be performed through random projections that change the entries of the tensor, dimensionality or size of modes, and/or the tensor order [2]. Matricization is a particular case of reshaping in which a tensor is unfolded or flattened to form a lower-order matrix by reordering the elements of the tensor. Thus, matricization can be thought as an operation that maps tensor entries into a matrix [11]. In mode-n matricization of Nth-order tensor, $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, the matrix is defined as $\mathbf{A}_{(n)} \in \mathbb{R}^{I_n \times I_1 I_2 \ldots I_{n-1} I_{n+1} \ldots I_n}$. This matrix has I_n rows and $I_1 I_2 \ldots I_{n-1} I_{n+1} \ldots I_n$ columns. The columns of the mode-n matricization are the mode-n fibers of tensor $\mathcal{A}$ [2]. Figure 2.3 illustrates how a mode-1 matricization works. In the figure the third-order tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is mode-1 matricized to reshape the tensor into the matrix $\mathbf{A}_{(1)} \in \mathbb{R}^{I_1 \times I_2 I_3}$. Vectorization is similar to matricization but in vectorization a tensor is converted into a vector.

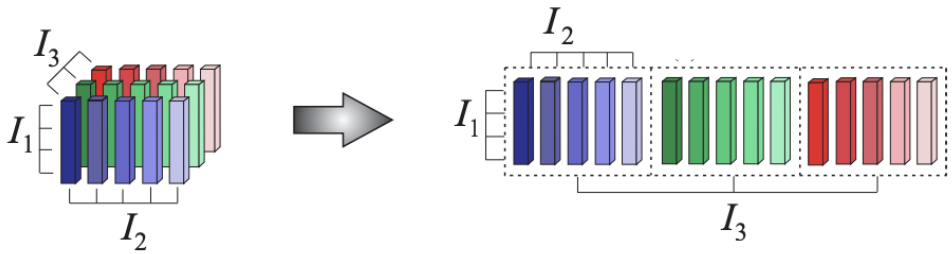


Figure 2.3: Mode-1 matricization of a third-order tensor to form a second-order matrix. Source: adapted from [2].

There exist several tensor products and some of those useful to this project are discussed next. Tensor product or outer product is a product between two tensors. For two tensors, $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathcal{B} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$, the tensor product between these tensors creates a new tensor, $\mathcal{C} = \mathcal{A} \circ \mathcal{B} \in \mathbb{R}^{I_1 \times \cdots \times I_N \times J_1 \times \cdots \times J_M}$. The tensor $\mathcal{C} \in \mathbb{R}^{I_1 \times \cdots \times I_N \times J_1 \times \cdots \times J_M}$ is $(N+M)$th-order and has entries $c_{i_1,\ldots,i_N,j_1\ldots,j_M} = a_{i_1,\ldots,i_N} b_{j_1,\ldots,j_M}$. For two nonzero vectors, $\mathbf{a} \in \mathbb{R}^I$ and $\mathbf{b} \in \mathbb{R}^J$, the tensor product is the outer product of these vectors that outputs a matrix, $\mathbf{C} = \mathbf{a} \circ \mathbf{b} = \mathbf{a}\mathbf{b}^\top \in \mathbb{R}^{I \times J}$. In a similar fashion, the tensor product of three nonzero vectors, $\mathbf{a} \in \mathbb{R}^I$, $\mathbf{b}^J$ and $\mathbf{c}^K$, produces a third-order tensor, $\mathcal{C} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \in \mathbb{R}^{I \times J \times K}$, that has entries $d_{ijk} = a_i b_j c_k$. When the tensor product is applied to nonzero vectors only, the resulting tensor is of rank-1 [2]. Also, a useful product is mode-n product of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and vector $\mathbf{b} \in \mathbb{R}^{I_n}$ which outputs a tensor, $\mathcal{C} = \mathcal{A} \,\bar{\times}_n\, \mathbf{b}$, $\mathcal{C} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N}$. The tensor $\mathcal{C}$ has entries $c_{i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N} = \sum_{i_n=1}^{I_n} a_{i_1,\ldots,i_{n-1},i_n,i_{n+1},\ldots,i_N} b_{i_n}$.

For two matrices, $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$, the Kronecker product is defined as $\mathbf{C} = \mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{IK \times JL}$ where

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B} & \cdots & a_{1,J}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{I,1}\mathbf{B} & \cdots & a_{I,J}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{IK \times JL}.$$

For two matrices, $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{I \times J}$, the Hadamard product is the element-wise multiplication of the entries of $\mathbf{A}$ and $\mathbf{B}$. It is defined as $\mathbf{C} = \mathbf{A} \odot \mathbf{B} \in \mathbb{R}^{I \times J}$:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{1,1}b_{1,1} & \cdots & a_{1,J}b_{1,J} \\ \vdots & \ddots & \vdots \\ a_{I,1}b_{I,1} & \cdots & a_{I,J}b_{I,J} \end{bmatrix} \in \mathbb{R}^{I \times J}.$$

Table 2.2 summarizes the tensor operations discussed above.

Table 2.2: Tensor operations summarized

$\mathbf{A}_{(n)} \in \mathbb{R}^{I_n \times I_1 \ldots I_{n-1} I_{n+1} \ldots I_N}$	mode-n matricization of $\mathcal{A} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$
$\mathcal{C} = \mathcal{A} \circ \mathcal{B}$	tensor or outer product of $\mathcal{A} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ and $\mathcal{B} \in \mathbb{R}^{J_1 \times \ldots \times J_M}$ yields $\mathcal{C} \in \mathbb{R}^{I_1 \times \ldots \times I_N \times J_1 \times \ldots \times J_M}$ with entries $c_{i_1 \ldots i_N j_1 \ldots j_M} = a_{i_1 \ldots i_N} b_{j_1 \ldots j_M}$
$\mathcal{C} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \ldots \circ \mathbf{a}^{(N)}$	tensor or outer product of vectors $\mathbf{a}^{(n)} \in \mathbb{R}^{I_n}$ $(n{=}1, \ldots, N)$ yields a rank-1 tensor $\mathcal{C} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ with entries $c_{i_1 i_2 \ldots i_N} = a^{(1)}_{i_1} a^{(2)}_{i_2} \ldots a^{(N)}_{i_N}$
$\mathcal{C} = \mathcal{A} \,\bar{\times}_n\, \mathbf{b} =$	mode-n product of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and vector $\mathbf{b} \in \mathbb{R}^{I_n}$ yields a tensor $\mathcal{C} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N}$ with entries $c_{i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N} = \sum_{i_n=1}^{I_n} a_{i_1, \ldots, i_{n-1}, i_n, i_{n+1}, \ldots, i_N} b_{i_n}$
$\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$	Kronecker product of $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$ yields $\mathbf{C} \in \mathbb{R}^{IK \times JL}$ with entries $c_{(i_1-1)J_1+j_1, (i_2-1)J_2+j_2} = a_{i_1 i_2} b_{j_1 j_2}$
$\mathbf{C} = \mathbf{A} \odot \mathbf{B}$	Hadamard product of $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{I \times J}$ yields $\mathbf{C} \in \mathbb{R}^{I \times J}$ with entries $c_{i,j} = a_{i,j} b_{i,j}$

2.1.3 Tensorization

Tensorisation is a key step in multi-linear analysis and refers to the creation of high-order tensors from lower-ordered structured datasets represented by vectors, matrices or low-order tensors. Tensorisation can be thought as the reverse process to the vectorization or matricization. Tensorisation can be roughly divided into the following four categories depending on how the high-order tensors are generated:

- *Natural tensor data.* Some of the data is naturally in the form of a tensor. For example, in HDTV, RGB colour images are generated as third-order tensors of size $1980 \times 1080 \times 3$ [11]. Generally speaking, videos naturally extend image data by including a time mode in addition to the pixel and spectral modes.

- *Experimental design.* Re-arrangement of data to reflect experimental setup, in other words, experiments that involve multiple variables such as trials, time and classes. For example, EEG recordings are best analysed when common modes of subjects, trials and conditions are combined together into a tensor [25].

- *Mathematical construction.* Additional features to represent the data are obtained through mathematics. For example, a ($channel \times time$) data matrix can be transformed into a ($channel \times time \times frequency$) third-order tensor via a time-frequency representation of the data.

- *Re-arrangement of lower-dimensional data.* Data can be reshaped or folded to transform lower-dimensional data into higher-order tensors. Re-arranging the lower-dimensional data is often done for data compression. For example, a climate time series stored into a vector can be segmented to a higher-order tensor (matrix) where each row/column corresponds to a particular day/year. Segmentation here means to extract and stack different segments from the data. [26], [27]

Figure 2.4 shows how a vector can be tensorised into higher-order tensors. First, the vector is tensorised into a matrix. Then the matrix is tensorised into a third-order tensor which is finally tensorised into a fourth-order matrix. Figure 2.5 shows an illustration that summarizes vectorization, matricization and tensorization. In the figure it can be seen how tensorization/matricizatio/vectorization can be used to reshape the structured datasets.

Some of the more sophisticated different tensorization methods presented in [13] are discussed next.

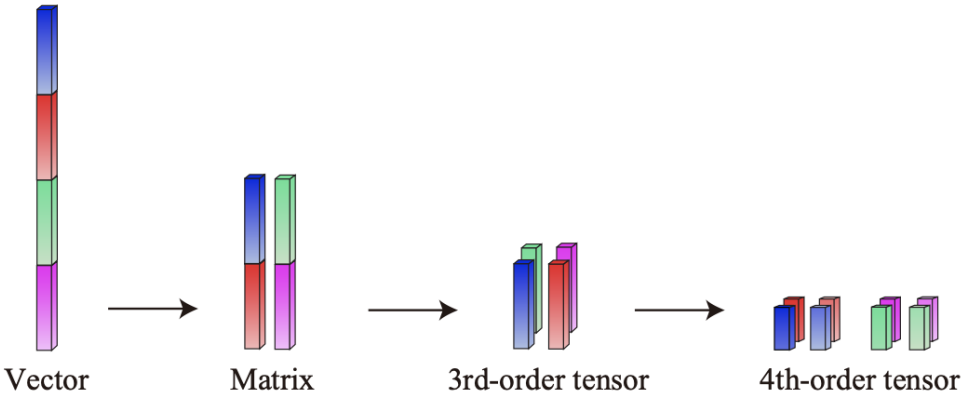


Figure 2.4: An example of tensorization of a vector $\mathbf{a} \in \mathbb{R}^{8K}$ into a matrix $\mathbf{A} \in \mathbb{R}^{4K \times 2}$. The matrix has then been then tensorized into a third-order tensor $\mathcal{A}_3 \in \mathbb{R}^{2K \times 2 \times 2}$. Finally, the third-order tensor has been tensorized into a fourth-order tensor $\mathcal{A}_4 \in \mathbb{R}^{K \times 2 \times 2 \times 2}$. Source: adapted from [2].

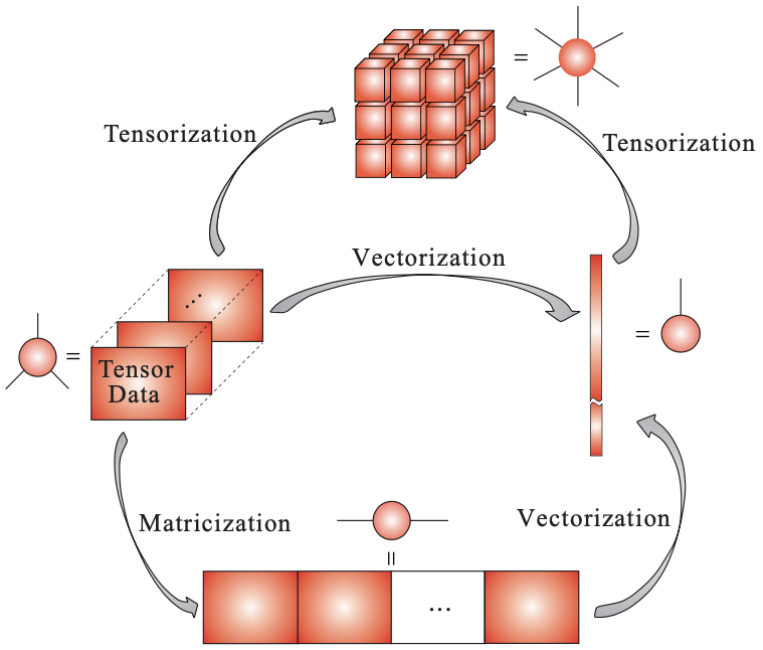


Figure 2.5: Reshaping operations performed on tensors. Tensors are represented by circles (nodes) with some number of outgoing lines. Each outgoing line from the circle (tensor or node) represents the indices of a specific node. Source: adapted from [2].

Reshaping or Folding

Reshaping or folding was briefly mentioned as one of the tensor operations in section 2.1.2 and will be summarized here along with some other tensorization techniques. Reshaping is the simplest tensorization method and preserves the original number of data entries as well as their sequential ordering. In reshaping a vector is rearranged to a matrix or tensor. This means that the memory space stays constant. A vector $\mathbf{y}$ of length $I_1 I_2 \dots I_N$ can be reshaped to a tensor $\mathcal{Y}$ of size $I_1 \times I_2 \times \cdots \times I_N$ such that

$$\mathcal{Y}(i_1, i_2, \dots, i_N) = \mathbf{y}(i) \quad \forall i \leq i_b \leq I_n, \tag{2.1}$$

where $i = 1 + \sum_{n=1}^{N} (i_n - 1) \prod_{k=1}^{n-1} I_k$ is a linear index of $(i_1, i_2, \dots)$. Reshaping can also be done as a reverse process such that a tensor $\mathcal{Y}$ is reformatted (vectorized) into a vector $\mathbf{y}$. Below is an example of reshaping a vector $\mathbf{y}$ of length L into a second-order tensor (or a matrix) of size

$I \times L/I$:

$$\mathbf{Y} = \begin{bmatrix} y(1) & y(I+1) & \dots & y(L-I+1) \\ y(2) & y(I+2) & \dots & y(L-I+2) \\ \vdots & \vdots & \ddots & \vdots \\ y(I) & y(2I) & \dots & y(L) \end{bmatrix} \tag{2.2}$$

In the case of higher-order reshaping, the method is applied several times. For example, a vector $\mathbf{y} \in \mathbb{R}^{I_1 I_2 \dots I_N}$ can be folded into an Nth-order tensor having size $I_1 \times I_2 \times \cdots \times I_N$.

Toeplitz Folding

A Toeplitz matrix is a structured matrix that has constant entries in each diagonal. It can be constructed from a vector $\mathbf{y}$ of length $L = I + J - 1$ such that the Toeplitz matrix of size $I \times J$ is defined as

$$\mathbf{Y} = \mathcal{T}_{I,J}(\mathbf{y}) = \begin{bmatrix} y(I) & y(I+1) & \dots & y(L) \\ y(I-1) & y(I) & \dots & y(L-1) \\ \vdots & \vdots & \ddots & \vdots \\ y(1) & y(2) & \dots & y(L-I+1) \end{bmatrix}. \tag{2.3}$$

In (2.3) the first column and first row of the matrix represent its entire generating vector $\mathbf{y}$. The representation of (2.3) is useful as the convolution between two vectors $\mathbf{x}$ and $\mathbf{y}$ of respective lengths I and $L > I$ is given by $\mathbf{z} = \mathbf{x} * \mathbf{y}$. The entries $\mathbf{z}_{I:L}$ are then such that $\mathbf{z}_{I:L} = \mathbf{Y}^T \mathbf{x}$. Therefore, the convolution can be simply computed through the Toeplitz matrix. Furthermore, an Nth-order Toeplitz tensor of size $I_1 \times I_2 \times \cdots \times I_N$ represented by $\mathcal{Y} = \mathcal{T}_{I_1,\dots,I_N}(\mathbf{y})$ is given by

$$\mathcal{Y}(i_1, \dots, i_{N-1}, i_N) = y(\bar{i}_1 + \cdots + \bar{i}_{N-1} + i_N), \tag{2.4}$$

where $\bar{i}_n = I_n - i_n$. [13]

Hankelization

Similar to Toeplitz matrix and tensor, Hankel matrix and tensor can also be used as linear operators in the convolution. An $I \times J$ Hankel matrix of a vector $\mathbf{y}$ of length $L = I + J - 1$ is defined as

$$\mathbf{Y} = \mathcal{H}_{I,J}(\mathbf{y}) = \begin{bmatrix} y(1) & y(2) & \dots & y(J) \\ y(2) & y(3) & \dots & y(J+1) \\ \vdots & \vdots & \ddots & \vdots \\ y(I) & y(I+1) & \dots & y(L) \end{bmatrix}. \tag{2.5}$$

Thus, an Nth-order Hankel tensor of size $I_1 \times I_2 \times \cdots \times I_N$ represented by $\mathcal{Y} = \mathcal{H}_{I_1,\dots,I_N}(\mathbf{y})$ is obtained from vector $\mathbf{y}$ of length $L = \sum_n I_n - N + 1$ in a way that its entries are defined as [28]

$$\mathcal{Y}(i_1, i_2, \dots i_N) = y(i_1 + i_2 + \cdots + i_N - N + 1). \tag{2.6}$$

Noteworthy in the case of Hankel tensors is that by fixing $N-2$ indices, in effect obtaining slices, the slices are also Hankel matrices. Also, a higher order Hankel tensor can be constructed from a lower order Hankel tensor by converting the lower order tensor's fibers to Hankel matrices. Moreover, in the case that the Hankel tensor has identical dimensions $I_n = I \ \forall n$, then the Hankel tensor is symmetric. [13]

It is obvious by comparing (2.3) and (2.5) that Hankel matrices/tensors and Toeplitz matrices/tensors are permuted versions of each other and the results can be transferred directly [26]. Therefore, out of the two tensorization methods, the focus will limited to Hankel matrices/tensors from this point onward.

Convolution Tensor

The entries of $(I, I+1, \dots, L)$ when convolving two vectors of respective lengths I and L, in effect, $\mathbf{x} * \mathbf{y}$, can be written as a tensor-vector product as follows:

$$[\mathbf{x} * \mathbf{y}]_{I:L} = \mathcal{C} \ \bar{\times}_1 \ \mathbf{x} \ \bar{\times}_3 \ \mathbf{y}. \tag{2.7}$$

In (2.7) $\mathcal{C}$ is a third-order tensor of size $I \times J \times L$, where $J = L - I + 1$. For $\mathcal{C}$ $(l - I)$-th, diagonal elements of l-th slices are ones and the remaining entries are zeros for $l = 1, 2, \ldots, L$. Slices $\mathcal{C}(:, :, l)$ for $l \leq I$ are then given as

$$\mathcal{C}(:, :, l) = \begin{bmatrix} 0 & & & 0 \\ & & & \\ 1 & & \ddots & \\ & \ddots & & \\ 0 & & 1 & 0 \end{bmatrix}. \tag{2.8}$$

The product between the convolution tensor $\mathcal{C}$ and vector $\mathbf{y}$ yields the Toeplitz matrix of the generating vector $\mathbf{y}$ of size $I \times J$:

$$\mathcal{T}_{I,J}(\mathbf{y}) = \mathcal{C} \; \bar{\times}_3 \; \mathbf{y}. \tag{2.9}$$

Moreover, the tensor-vector product $\mathcal{C} \; \bar{\times}_1 \; \mathbf{x}$ yields a Toeplitz matrix of the generating vector $[\mathbf{0}_{L-I}^T, \mathbf{x}^T, \mathbf{0}_{J-1}^T]^T$:

$$\mathcal{C} \; \bar{\times}_1 \; \mathbf{x} = \mathcal{T}_{L,J}([\mathbf{0}_{L-I}^T, \mathbf{x}^T, \mathbf{0}_{J-1}^T]^T). \tag{2.10}$$

Consider the case of a convolution of $(N - 1)$ vectors, $\mathbf{x}_1, \ldots, \mathbf{x}_{N-1}$ having respective lengths $I_1, \ldots, I_{N-1}$ and a vector $\mathbf{y}$ having length L, in effect

$$\mathbf{z} = \mathbf{x}_1 * \mathbf{x}_2 * \cdots * \mathbf{x}_{N-1} * \mathbf{y}. \tag{2.11}$$

Then, the entries of $\mathbf{z}$ can be expressed as a multilinear product of a convolution tensor $\mathcal{C}$ of $(N + 1)$th-order and size $I_1 \times I_2 \times \cdots \times I_N \times L$, $I_N = L - \sum_{n=1}^{N-1} I_n + N - 1$ and the N input vectors:

$$\mathbf{z}_{L-I_N+1:L} = \mathcal{C} \; \bar{\times}_1 \; \mathbf{x}_1 \; \bar{\times}_2 \; \mathbf{x}_2 \ldots \bar{\times}_{N-1} \; \mathbf{x}_{N-1} \; \bar{\times}_{N+1} \; \mathbf{y}. \tag{2.12}$$

In (2.12) the entries of $\mathcal{C}$ are zeros apart from the entries located at $(i_1, i_2, \ldots, i_{N+1})$ such that

$$\sum_{n=1}^{N-1} \bar{i}_n + i_N - i_{N+1} = 0, \tag{2.13}$$

where $\bar{i}_n = I_n - i_n,\ i_n = 1, 2, \ldots, I_n$.

Finally, considering the tensor product $\mathcal{C}\ \bar{\times}_{N+1}\ \mathbf{y}$, one obtains the Toeplitz tensor of the generating vector $\mathbf{y}$:

$$\mathcal{C}\ \bar{\times}_{N+1}\ \mathbf{y} = \mathcal{T}_{I_1,\ldots,I_N}(\mathbf{y}). \tag{2.14}$$

[13]

Tensorization by Means of Löwner Matrix

For a vector $\mathbf{v} \in \mathbb{R}^{I+J}$, a Löwner matrix is formed from a function $f(t)$ that is sampled at $(I + J)$ distinct points $\{x_1, \ldots, x_I, y_1, \ldots, y_J\}$. Then

$$\mathbf{v} = [f(x_1), \ldots, f(x_I), f(y_1), \ldots, f(y_J)]^T \in \mathbb{R}^{I+J} \tag{2.15}$$

from which it can be seen that the entries of $\mathbf{v}$ are partitioned into two disjoint sets, $\{f(x_i)\}_{i=1}^{I}$ and $\{f(y_j)\}_{j=1}^{J}$. Conversion of the vector $\mathbf{v}$ into the Löwner matrix, $\mathbf{L} \in \mathbb{R}^{I\times J}$, is defined by

$$\mathbf{L} = \left[\frac{f(x_i) - f(y_j)}{x_i - y_j}\right]_{ij} \in \mathbb{R}^{I\times J}. \tag{2.16}$$

Löwner matrices are a useful tool in fitting a model to data through a rational (Pade form) approximation, in effect, $f(x) = A(x)/B(x)$. Considering rational approximations as transfer functions, rational approximations are more powerful than polynomial approximations as they are able to model discontinuities and spiky data. Rank of the Löwner matrix gives the optimal order of such a rational approximation. In the case of tensors, one is then able to construct a model of the original dataset which is amenable to higher-order tensor representation while having minimal computational complexity and whose accuracy is determined by the rank of the Löwner matrix. Consider the following example of Löwner folding to a vector

$\mathbf{x} = [1/3, 1/4, 1/5, 1/6, 1/8, 1/9, 1/10]$. Applying Löwner folding to the vector yields

$$\begin{bmatrix} \frac{1/3-1/8}{3-8} & \frac{1/3-1/9}{3-9} & \frac{1/3-1/10}{3-10} \\ \frac{1/4-1/8}{4-8} & \frac{1/4-1/9}{4-9} & \frac{1/4-1/10}{4-10} \\ \frac{1/5-1/8}{5-8} & \frac{1/5-1/9}{5-9} & \frac{1/5-1/10}{5-10} \\ \frac{1/6-1/8}{6-8} & \frac{1/6-1/9}{6-9} & \frac{1/6-1/10}{6-10} \end{bmatrix} = - \begin{bmatrix} 1/3 \\ 1/4 \\ 1/5 \\ 1/6 \end{bmatrix} \begin{bmatrix} 1/8 & 1/9 & 1/10 \end{bmatrix}.$$

[13] More applications of tensorization by means of Löwner matrix are presented in [29].

Tensor Structures in Multivariate Polynomial Regression (MPR)

Consider a case of fitting a curve to data with two independent variables x_1 and x_2:

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2, \tag{2.17}$$

where w_{12} represents the strength of interaction between x_1 and x_2. To describe more complex functional behaviours, the model can have more terms, for example, x_1^2 and $x_1 x_2^2$. Thus, the full quadratic polynomial regression model for two independent variables x_1 and x_2 is given by

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2 + w_{11} x_1^2 + w_{22} x_2^2 + w_{112} x_1^2 x_2 + w_{122} x_1 x_2^2 + w_{1122} x_1^2 x_2^2. \tag{2.18}$$

Using tensor representation of the system weights, (2.17) can be rewritten as

$$y = \begin{bmatrix} 1 & x_1 \end{bmatrix} \begin{bmatrix} w_0 & w_2 \\ w_1 & w_{12} \end{bmatrix} \begin{bmatrix} 1 \\ x_2 \end{bmatrix}. \tag{2.19}$$

The full quadratic regression model in (2.18) can be rewritten using tensor representation of

the system weights as

$$y = \begin{bmatrix} 1 & x_1 & x_1^2 \end{bmatrix} \begin{bmatrix} w_0 & w_2 & w_{22} \\ w_1 & w_{12} & w_{122} \\ w_{11} & w_{112} & w_{1122} \end{bmatrix} \begin{bmatrix} 1 \\ x_2 \\ x_2^2 \end{bmatrix} \tag{2.20}$$

or as a tensor-vector product representation as

$$y = \mathcal{W} \; \bar{\times}_1 \begin{bmatrix} 1 \\ x_1 \end{bmatrix} \bar{\times}_2 \begin{bmatrix} 1 \\ x_1 \end{bmatrix} \bar{\times}_3 \begin{bmatrix} 1 \\ x_2 \end{bmatrix} \bar{\times}_4 \begin{bmatrix} 1 \\ x_2 \end{bmatrix}, \tag{2.21}$$

where $\mathcal{W}$ is fourth-order tensor of size $2 \times 2 \times 2 \times 2$ given by

$$\mathcal{W}(:,:,1,1) = \begin{bmatrix} w_0 & \frac{1}{2}w_1 \\ \frac{1}{2}w_1 & w_{11} \end{bmatrix}, \; \mathcal{W}(:,:,2,2) = \begin{bmatrix} w_{22} & \frac{1}{2}w_{122} \\ \frac{1}{2}w_{122} & w_{1122} \end{bmatrix},$$
$$\mathcal{W}(:,:,1,2) = \mathcal{W}(:,:,2,1) = \frac{1}{2} \begin{bmatrix} w_2 & \frac{1}{2}w_{12} \\ \frac{1}{2}w_{12} & w_{112} \end{bmatrix}.$$

For a generalised system that consists of N independent variables, $x_1, \ldots, x_N$, the MPR can be written as a tensor-vector product as [30]

$$\begin{aligned} y &= \sum_{i_1=0}^{N} \sum_{i_2=0}^{N} \cdots \sum_{i_N=0}^{N} w_{i_1,i_2,\ldots,i_N} x_1^{i_1} x_2^{i_2} \ldots x_N^{i_N} \\ &= \mathcal{W} \; \bar{\times}_1 \; \mathrm{V}_N(x_1) \; \bar{\times}_2 \; \mathrm{V}_N(x_2) \ldots \bar{\times}_N \; \mathrm{V}_N(x_N). \end{aligned} \tag{2.22}$$

In (2.22) $\mathcal{W}$ is an Nth-order tensor of size $(N+1) \times (N+1) \times \cdots \times (N+1)$ and $\mathrm{V}_N(x)$ is length $(N+1)$ Vandermonde vector of x given by

$$\mathrm{V}_N(x) = \begin{bmatrix} 1 & x & x^2 & \ldots & x^N \end{bmatrix}^T. \tag{2.23}$$

MPR has been used in several applications as its able to model any smooth, continuous nonlinear

input-output system [31]. The drawback of the MPR is that since the number of parameters in the model in (2.22) shows exponential growth with the number of variables, N, the MPR needs a vast amount of data to yield a good model. This means that the MPR is computationally intensive in a raw tensor format and not suitable for very high-dimensional data. [13]

Tensor Structures in Vector-variate Regression

The MPR in (2.22) works with scalar data only. However, the model can easily be extended for vectors or tensors in the case the observations are in those formats. This time consider the case of two independent vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. The nonlinear mapping from the input to the output, in effect, $y = h(\mathbf{x}_1, \mathbf{x}_2)$ can be linearly approximated as

$$\begin{aligned} y = h(\mathbf{x}_1, \mathbf{x}_2) &= w_0 + \mathbf{w}_1^T \mathbf{x}_1 + \mathbf{w}_2^T \mathbf{x}_2 + \mathbf{x}_1^T \mathbf{W}_{12} \mathbf{x}_2 \\ &= \begin{bmatrix} 1 & \mathbf{x}_1^T \end{bmatrix} \begin{bmatrix} w_0 & \mathbf{w}_2^T \\ \mathbf{w}_1 & \mathbf{W}_{12} \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{x}_2 \end{bmatrix}. \end{aligned} \tag{2.24}$$

The full quadratic regression model is given by

$$\begin{aligned} y = h(\mathbf{x}_1, \mathbf{x}_2) &= w_0 + \mathbf{w}_1^T \mathbf{x}_1 + \mathbf{w}_2^T \mathbf{x}_2 + \mathbf{x}_1^T \mathbf{W}_{12} \mathbf{x}_2 + \mathbf{x}_1^T \mathbf{W}_{11} \mathbf{x}_1 + \mathbf{x}_2^T \mathbf{W}_{22} \mathbf{x}_2 \\ &\quad + \mathcal{W}_{112} \,\bar{\times}_1\, \mathbf{x}_1 \,\bar{\times}_2\, \mathbf{x}_1 \,\bar{\times}_3\, \mathbf{x}_2 + \mathcal{W}_{122} \,\bar{\times}_1\, \mathbf{x}_1 \,\bar{\times}_2\, \mathbf{x}_2 \,\bar{\times}_3\, \mathbf{x}_2 \\ &\quad + \mathcal{W}_{1122} \,\bar{\times}_1\, \mathbf{x}_1 \,\bar{\times}_2\, \mathbf{x}_1 \,\bar{\times}_3\, \mathbf{x}_2 \,\bar{\times}_4\, \mathbf{x}_2 \\ &= \begin{bmatrix} 1 & \mathbf{x}_1^T & (\mathbf{x}_1 \otimes \mathbf{x}_1)^T \end{bmatrix} \mathbf{W} \begin{bmatrix} 1 \\ \mathbf{x}_2 \\ \mathbf{x}_2 \otimes \mathbf{x}_2 \end{bmatrix}, \end{aligned} \tag{2.25}$$

where the matrix $\mathbf{W}$ is given by

$$\mathbf{W} = \begin{bmatrix} w_0 & \mathbf{w}_2^T & \mathrm{vec}(\mathbf{W}_{22})^T \\ \mathbf{w}_1 & \mathbf{W}_{12} & [\mathcal{W}_{122}]_{(1)} \\ \mathrm{vec}(\mathbf{W}_{11}) & [\mathcal{W}_{112}]_{(1,2)} & [\mathcal{W}_{1122}]_{(1,2)} \end{bmatrix} \tag{2.26}$$

In (2.26), $[\mathcal{W}_{112}]_{(1,2)}$ is the mode-$(1,2)$ unfolding of the tensor $\mathcal{W}_{112}$. The above model can be expressed through a tensor-vector product expression, similar to (2.21), as

$$y = \mathcal{W} \bar{\times}_1 \begin{bmatrix} 1 \\ \mathbf{x}_1 \end{bmatrix} \bar{\times}_2 \begin{bmatrix} 1 \\ \mathbf{x}_1 \end{bmatrix} \bar{\times}_3 \begin{bmatrix} 1 \\ \mathbf{x}_2 \end{bmatrix} \bar{\times}_4 \begin{bmatrix} 1 \\ \mathbf{x}_2 \end{bmatrix}, \tag{2.27}$$

where $\mathcal{W}$ is a fourth-order tensor.

Vector-variate regression with N input vectors, $\mathbf{x}_n$ of lengths I_n, is as follows:

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = w_0 + \sum_{d=1}^{N^2} \sum_{i_1, i_2, \ldots, i_d = 1}^{N} \mathcal{W}_{i_1, i_2, \ldots, i_d} \bar{\times} (\mathbf{x}_{i_1} \circ \mathbf{x}_{i_2} \circ \cdots \circ \mathbf{x}_{i_d}), \tag{2.28}$$

where $\bar{\times}$ is the inner product between two tensors and the tensors $\mathcal{W}_{i_1, \ldots, i_d}$ are d-th order and have size $I_{i_1} \times I_{i_2} \times \cdots \times I_{i_d}$, $d = 1, \ldots, N^2$. Moreover, the representation of (2.28) as a tensor-vector product of an Nth-order tensor having size $J_1 \times J_2 \times \cdots \times J_N$, where $J_n = \frac{I_n^{N+1} - 1}{I_n - 1}$, comprising all the weights is defined as

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \mathcal{W} \bar{\times}_1 \mathbf{v}_N(\mathbf{x}_1) \bar{\times}_2 \mathbf{v}_N(\mathbf{x}_2) \ldots \bar{\times}_N \mathbf{v}_N(\mathbf{x}_N), \tag{2.29}$$

where

$$\mathbf{v}_N(\mathbf{x}) = \begin{bmatrix} 1 & \mathbf{x}^T & (\mathbf{x} \otimes \mathbf{x})^T & \ldots & (\mathbf{x} \otimes \cdots \otimes \mathbf{x})^T \end{bmatrix}^T. \tag{2.30}$$

[13]

2.1.4 Tensor Decomposition Methods

TD methods are valuable as they provide ways to compress the data and as such allow for extraction of the data from tensors and make the data more interpretable. Furthermore, TD methods allow one to solve many high-dimensional data and machine learning problems without suffering from the curse of dimensionality. [1] TD methods' major application lies in their advantage in the analysis of data exhibiting not only large volume but also very high variety. As such, they have been used in diverse disciplines including psychometrics, chemometrics,

biometric, quantum physics / information, quantum qhemistry, signal and image processing, machine learning, and brain science, among other disciplines. [2]

TD methods work by decomposing principally data tensors into factor matrices. Therefore, one is able to have highly compressed and distributed formats representation of large-scale data. [2] Tucker decomposition and TT decomposition are popular TD methods and as such explained below. In this project TT decomposition is used in the empirical work and therefore this work could naturally be extended to explore the use of Tucker decomposition instead.

Tucker Decomposition

Tucker decomposition of Nth-order tensor, $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, provides a more general factorization of $\mathcal{X}$ into a relatively small size core tensor and factor matrices [2]:

$$\begin{aligned}\mathcal{X} &\approx \sum_{r_1=1}^{R_1} \cdots \sum_{r_N=1}^{R_N} g_{r_1 r_2 \ldots r_N} (\mathbf{b}_{r_1}^{(1)} \circ \mathbf{b}_{r_2}^{(2)} \circ \cdots \circ \mathbf{b}_{r_N}^{(N)}) \\ &= \mathcal{G} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \cdots \times_N \mathbf{B}^{(N)} \\ &= [[\mathcal{G}; \mathbf{B}^{(1)}, \mathbf{B}^{(2)}, \ldots, \mathbf{B}^{(N)}]], \end{aligned} \tag{2.31}$$

where $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ represents the core tensor, and $\mathbf{B}^{(n)} = [\mathbf{b}_1^{(n)}, \mathbf{b}_2^{(n)}, \ldots, \mathbf{b}_{R_n}^{(n)}] \in \mathbb{R}^{I_n \times R_n}$ are the mode-n factor matrices. Figure 2.6 shows an illustration of Tucker decomposition of a third-order tensor.

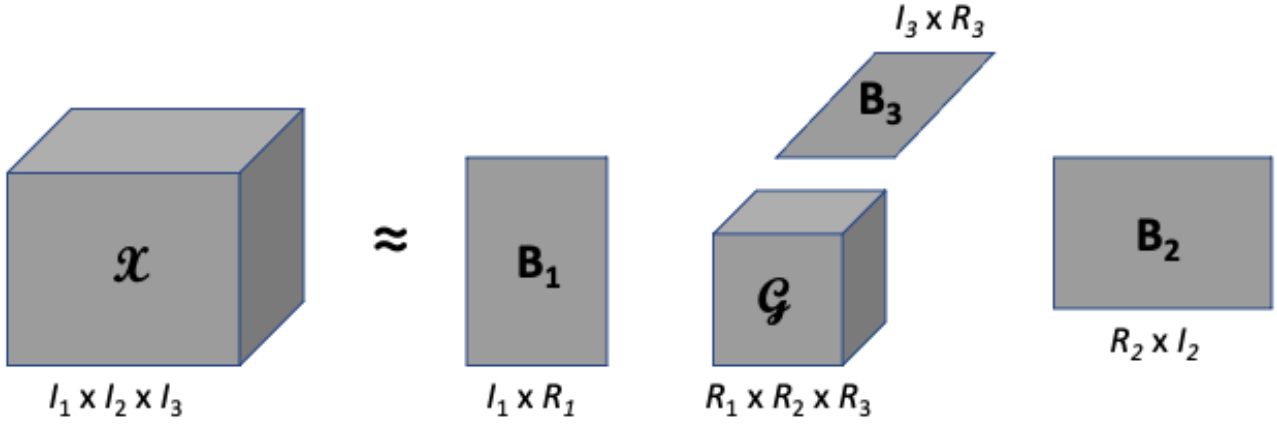


Figure 2.6: Illustration of Tucker decomposition of a third-order tensor.

However, the drawback of Tucker decomposition is that it is only suitable for "small" dimensions, particularly for a three-dimensional case. When the dimensions become very high, Tucker decomposition is not suitable any more. [3]

Tensor-Train Decomposition

TT decomposition provides a solution to the limitation of Tucker decomposition. In TT decomposition, dth-order tensor $\mathcal{A} \in \mathbb{R}^{p_1 \times p_2 \times \cdots \times p_d}$ is represented through multiplication of cores $\mathcal{G}_1, \ldots, \mathcal{G}_d$ [3]:

$$\mathcal{A}(i_1, i_2, \ldots, i_d) = \mathcal{G}_1(i_1)\mathcal{G}_2(i_2)\ldots\mathcal{G}_d(i_d), \tag{2.32}$$

where the cores are third-order tensors and have size (p_k, r_{k-1}, r_k), $i_k \in [1, p_k]$ $\forall k \in [1, d]$ and $r_0 = r_d = 1$. Figure 2.7 illustrates TT decomposition.

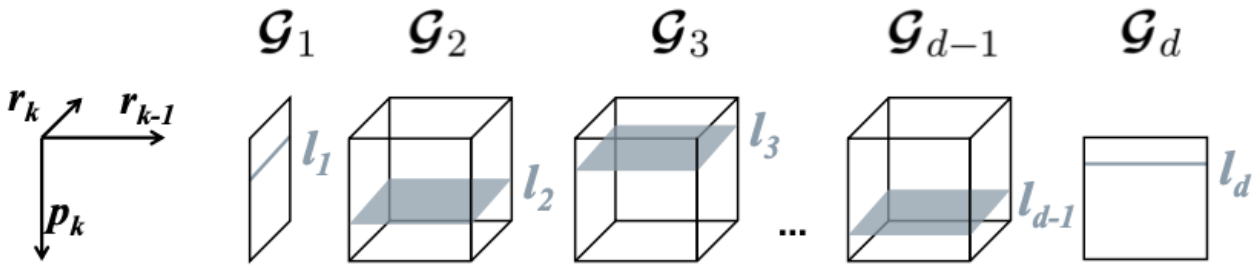


Figure 2.7: Illustration of TT decomposition. Source: adapted from [3].

2.2 Forecasting Methods

Forecasting methods used in the project are presented in this section. More specifically, NN, ARIMA and GARCH models are explained. ARIMA and GARCH models are detailed as this project could be extended to those models as future work.

2.2.1 Neural Network

NNs are a way of doing ML in which a computer learns to perform a task such as time series forecasting by analyzing training examples [32]. NNs provide a wide research field and as such are constantly evolving. To narrow down the scope, this section introduces a vanilla fully connected NN as well as a RNN. Furthermore, activation functions, loss functions and training NNs are discussed.

Fully Connected Neural Network

As the name of NN implies, the most fundamental element in NNs is a neuron which is illustrated in figure 2.8. Neuron takes an input, say vector $\mathbf{x}$. Then, $\mathbf{x}$ is multiplied with weights vector $\boldsymbol{\theta}$ and a bias b is added to the result. Finally, an activation function ϕ is applied to result to arrive at $\hat{y}$. This procedure is summarized as follows:

$$\hat{y} = \phi(\sum_{i=1} \theta_1 x_1 + b) = \phi(\boldsymbol{\theta}^T \mathbf{x} + b). \tag{2.33}$$

It is possible to connect multiple neurons in parallel for a particular layer such that each neuron learns to detect something different from the input. Furthermore, to arrive at a NN, multiple layers consisting of multiple neurons are connected with each other. This is shown in figure

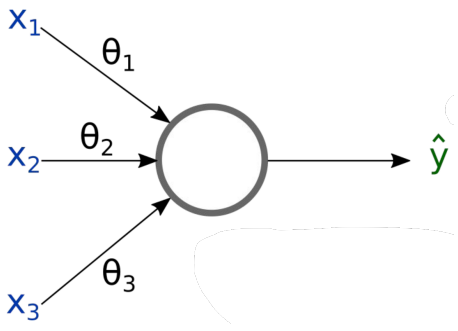


Figure 2.8: Neuron visualized for an input length of three. Source: adapted from [4].

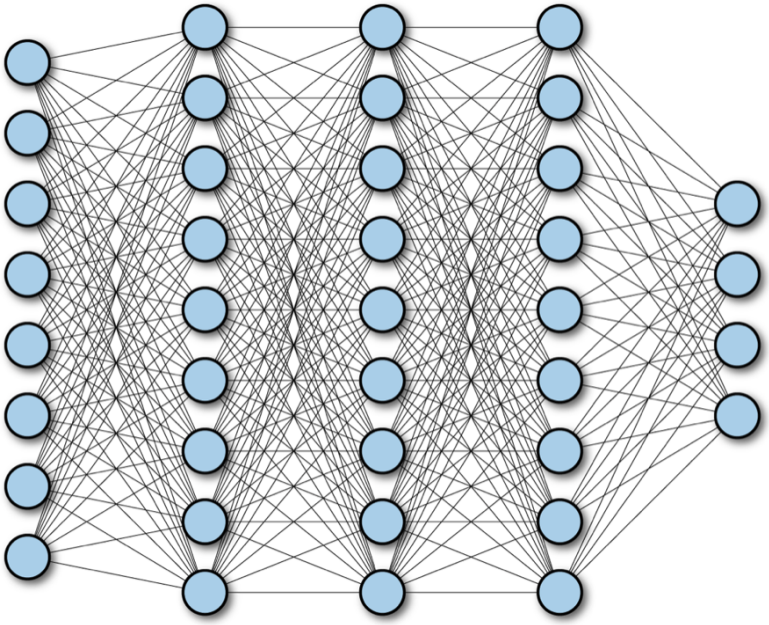


Figure 2.9: Visualization of fully connected NN. Source: adapted from [5].

2.9. The output $\hat{\mathbf{y}}$ of a single fully connected layer is defined as follows:

$$\hat{\mathbf{y}} = \phi(\mathbf{\Theta}\mathbf{x} + \mathbf{b}), \tag{2.34}$$

where $\mathbf{\Theta} \in \mathbb{R}^{N \times M}$ is a weights matrix and $\mathbf{b}$ and $\hat{\mathbf{y}}$ represent bias and output vectors, respectively.

Recurrent Neural Network

The problem with standard NNs is that they suffer from linking previous data to new data, in effect, capturing temporal dependencies in the data. Financial data is temporal where current samples are affected by previous samples. Therefore, more advanced NNs are needed to model time series data. RNNs are suited for processing sequential data such as time series, video data and speech data, and if trained well, can model dependencies within a sequence of arbitrary length [6]. This is because RNNs have memory units that store information from previously seen data meaning that when predicting outputs, not only the current input is used but also the previous inputs are taken into account. Figure 2.10 illustrates a simple RNN.

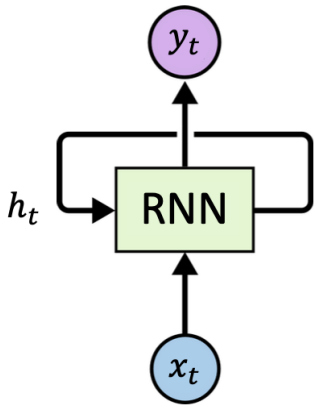


Figure 2.10: Simple RNN. Source: adapted from [6].

A simple RNN takes as an input vector $\mathbf{x}$ and learns the mapping from the inputs to outputs $\mathbf{y}$ by computing the following [6]:

$$\mathbf{h}_t = \phi_h(W_h\mathbf{h}_{t-1} + W_x\mathbf{x}_t + \mathbf{b}_h), \tag{2.35}$$

$$\mathbf{y}_t = \phi_y(W_y\mathbf{h}_t + \mathbf{b}_y), \tag{2.36}$$

where the network parameters are $\boldsymbol{\theta} = \{W_h, W_x, W_y, \mathbf{b}_h, \mathbf{b}_y\}$ and ϕ_h and ϕ_y represent non-linear activation functions for the hidden state, $\mathbf{h}_t$, and the output, $\mathbf{y}_t$, respectively. The convention is to set $\mathbf{h}_0 = \mathbf{0}$ for $t = 1$ such that $\mathbf{h}_1 = \phi_h(W_x\mathbf{x}_1 + \mathbf{b}_h)$. [6] As it can be seen from (2.35), the hidden states $\mathbf{h}_t$ store information and propagate in the network. Commonly in NN training is spoken about back-propagation when updating the parameters (weights). In the case of RNNs, it is instead talked about back-propagation through time (BPTT). Figure 2.11 illustrates BPTT.

There exist more sophisticated variations of RNNs such as gated recurrent units (GRUs) and long-short term memory units (LSTMs). These tackle the problems associated with vanishing/exploding gradients of RNNs. However, this project focuses solely on RNNs but future work could be extended to tensor GRUs and LSTMs.

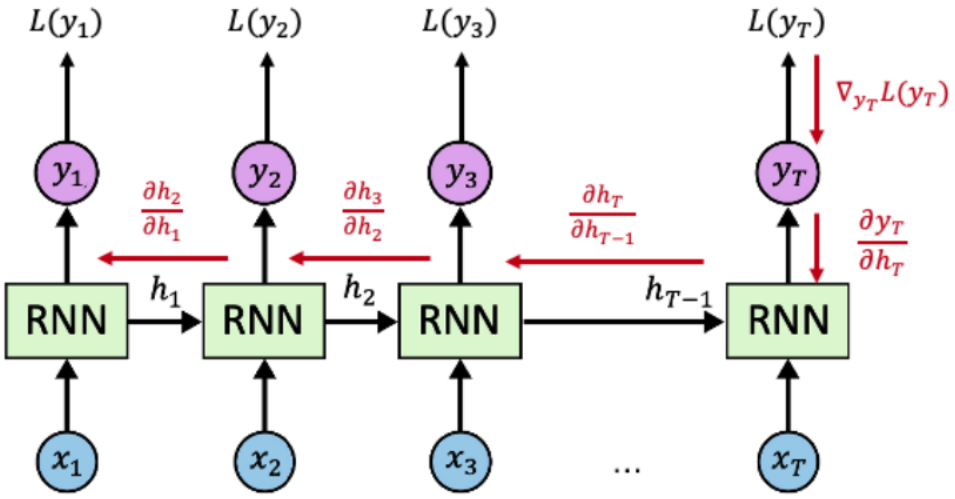


Figure 2.11: Visualization of BPTT in RNN. The black arrows show forward pass computations of the hidden state. The red arrows show the gradient back-propagation in order to compute $\nabla_{W_h} L(y_t)$, where L refers to a loss function. Source: adapted from [6].

Activation Functions

Activation functions enable NNs to learn more complex patterns in the data and thus, for example, model non-linear functions. There exist several activation functions such as `ReLU`, `sigmoid`, `tanh` and `softmax` which are used depending on the application. In this project `tanh` and `softmax` are used and they are defined as follows:

$$\text{output}_{\text{tanh}} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.37}$$

$$\text{output}_{\text{softmax}} = \frac{e^{x_i}}{\sum_j e^{x_j}}. \tag{2.38}$$

Therefore, `tanh` is a nonlinear activation function that is bound to range $(-1, 1)$ which means that the activations will not explode. `softmax` rescales the elements of an N-dimensional input tensor such that the outputs of the N-dimensional output tensor are in the range $[0, 1]$ and sum up to 1. [33]

Training

NNs are trained with the intention to learn patterns in the data by analyzing training examples. This is done by using an objective function such as minimizing a measure of error and finding a set of optimal weights. The objective function could be, for example, cross-entropy loss which is used in this project. Cross-entropy loss is often used in multi-class classification problems and it is defined as [33]

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{N} \mathbf{y}_i \log(\hat{\mathbf{y}}_i). \tag{2.39}$$

Therefore, the training problem can be generalized as follows [34]:

$$\boldsymbol{\Theta}^* = \text{argmin}_{\boldsymbol{\Theta}}\ \mathbb{E}[L(\mathbf{y}, \hat{\mathbf{y}})]. \tag{2.40}$$

To find the optimal weights, stochastic gradient descent (SGD) can be used. It is an iterative algorithm that updates the weights according to the objective function as follows [34]:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \frac{\nabla Q(\boldsymbol{\theta})}{||\nabla Q(\boldsymbol{\theta})||}, \tag{2.41}$$

where α is the learning rate and $Q(\boldsymbol{\theta}) = \mathbb{E}[L(\mathbf{y}, \hat{\mathbf{y}})]$ represents the average loss for the training data batch.

2.2.2 ARIMA

ARIMA model is a popular method in time series forecasting due to its flexibility, simplicity and statistical properties [35]. It is a stochastic model that uses historical data to train the model which can then be used to forecast future values and patterns. ARIMA model effectively combines the autoregressive (AR) model and the moving average (MA) model and uses the integrated (I) component. AR model accounts for the patterns between any one time period and the previous periods [36]. Therefore, an AR process is a stochastic difference equation in which the current value of a series is linearly related to its past values with additive stochastic shock [37]. MA model which is also known as the error feedback term measures adaptation of

new forecasts to prior forecast errors [36]. This means that the current value of the observed series is expressed as a function of current and lagged unobservable shocks [38]. The integrated component on the other hand connotes a trend in the data [36]. Essentially, it is used to ensure that the time series is stationary. This means that the statistical properties such as mean and variance of the time series are constant over time. To make time series stationary, ARIMA models use differencing [39]. Differencing can be used iteratively until the time series becomes stationary. For example, in first order differencing the previous data point is subtracted from the current data point. Furthermore, when using logarithmic prices, the difference between any two consecutive data points, essentially the logarithmic return, represents first order differencing. [9]

ARMA(p,q) model is defined as [8]

$$\mathrm{x}_t = \sum_{i=1}^{p} a_i \mathrm{x}_{t-i} + \sum_{i=1}^{q} b_i \epsilon_{t-i} + \epsilon_t, \tag{2.42}$$

where x_t is the actual data value at time t and $\{a_i\}_{i=1}^{p}$ and $\{b_i\}_{i=1}^{q}$ are the AR and MA coefficients, respectively. The ARMA model is suitable if the time series used is already stationary. If not, then ARIMA(p,d,q) model is more suitable. ARIMA(p,d,q) is defined as [8]

$$\Delta^d \mathrm{x}_t = \sum_{i=1}^{p} a_i \Delta^d \mathrm{x}_{t-i} + \sum_{i=1}^{q} b_i \epsilon_{t-i} + \epsilon_t, \tag{2.43}$$

where $\Delta^d \mathrm{x}_t$ refers to order-d differencing of x_t to make the time series stationary. If the time series is already stationary, then $d = 0$ and (2.42) and (2.43) are identical.

ARIMA model defined in (2.43) can be extended to tensors. The classical ARIMA model was generalized for tensors and Tucker decomposition was incorporated into the model in [8]. This was achieved by first applying Hankelization to the input tensor $\mathcal{X}$ to arrive at a block Hankel tensor $\hat{\mathcal{X}}_t$. After that order-d differencing was computed to get $\{\Delta^d \hat{\mathcal{X}}_t\}_{t=d}^{\hat{T}}$ for which Tucker decomposition was employed by projecting it to core tensors $\{\Delta^d \hat{\mathcal{G}}_t\}$. A forecasting model was then trained directly using the core tensors instead of whole tensors. In order to retain temporal correlations among the core tensors, a generalization of (p, d, q)-order ARIMA model to tensor

form was implemented and used to connect the current core tensor $\Delta^d\hat{\mathcal{G}}_t$ and the previous core tensors $\Delta^d\hat{\mathcal{G}}_{t-1}, \Delta^d\hat{\mathcal{G}}_{t-2}, \ldots, \Delta^d\hat{\mathcal{G}}_{t-p}$:

$$\Delta^d\hat{\mathcal{G}}_t = \sum_{i=1}^{p} \alpha_i \Delta^d\hat{\mathcal{G}}_{t-i} + \sum_{i=1}^{q} \beta_i \hat{\boldsymbol{\mathcal{E}}}_{t-i} + \hat{\boldsymbol{\mathcal{E}}}_t. \tag{2.44}$$

[8]

2.2.3 GARCH

ARIMA models are used to predict the future values, for example, the returns of a particular stock, given the past returns such that the conditional variance stays constant. However, the constant conditional variance may not be satisfactory if one expects, say, more volatility for any particular stock in the near future. For example, if the recent daily returns for the stock have been surprisingly volatile, one would expect more volatile returns for tomorrow as well. Since ARIMA model has constant conditional variance, it is not able to capture this type of volatile behaviour. GARCH models on the other hand have randomly varying volatility. Therefore, GARCH models are able to model the nonconstant volatility. [40]

Let ϵ_t represent white noise with unit variance. a_t is an ARCH(p) process if

$$a_t = \sigma_t \epsilon_t, \tag{2.45}$$

where

$$\sigma_t = \sqrt{\omega + \sum_{i=1}^{p} \alpha_i a_{t-i}^2} \tag{2.46}$$

represents the conditional standard deviation of a_t. Thus, for ARCH(p) model, the variance for σ^2 at time t is conditional on the previous p observations:

$$\mathrm{Var}(a_t | a_{t-1}, \ldots, a_{t-p}) = \sigma_t^2 = \omega + \alpha_1 a_{t-1}^2 + \cdots + \alpha_p a_{t-p}^2. \tag{2.47}$$

In ARCH(p) models the conditional standard deviation process has high-frequency oscillations

with high volatility that come in short bursts. This is a deficiency of the model. GARCH models offer a solution to this as they permit a wider range of behavior such as more persistent periods of high or low volatility. This is because past values of the σ_t process are used to define the new σ_t. GARCH(p, q) model is defined as

$$a_t = \sigma_t \epsilon_t, \tag{2.48}$$

where

$$\sigma_t = \sqrt{\omega + \sum_{i=1}^{p} \alpha_i a_{t-i}^2 + \sum_{i=1}^{q} \beta_i \sigma_{t-i}^2}. \tag{2.49}$$

Note that the a_t process is uncorrelated with a stationary mean and variance. Moreover, a_t^2 has an autocorrelation function (ACF) like an ARIMA process. [40]

For a more exhaustive review on ARCH and GARCH as well as ARIMA/GARCH models, please refer to [40]. [41] extends to multivariate GARCH models.

There was no related work found regarding the use of tensors in GARCH models. As such, to account for heteroscedasticity which is a ubiquitous phenomenon in financial variables while reaping the benefits from the use of tensors, one could extend this work to tensor GARCH models.

2.3 Financial Background

This project deals with financial forecasting and thus a brief overview of different financial asset classes as well as indices and financial instruments is provided next.

Asset class refers to a grouping of comparable financial securities having similar characteristics and that are subject to same regulations and laws. Depending on the convention, there are anywhere from three up to at least nine different asset classes. However, historically speaking, the three main asset classes include equities (stocks), fixed income (bonds), and cash

equivalents or money market instruments. Today, real estate, commodities and even emerging cryptocurrencies are included in the asset class mix. [42]

Elaborating a bit more on the asset classes, equities represent ownership in public companies. Fixed income assets usually have a set interest rate paid to the investor over a given period after which the initial investment is paid back. Cash equivalents are securities intended for short-term investing and as such they are highly liquid and often have high credit quality [43]. Commodities refer to natural resources such as raw materials, agricultural products and metals. Since the prices of the various individual securities within each asset class may vary greatly, indices are used to track the different asset classes. Indices give investors a better idea of how different markets perform. For example in equities, there are several different stock indices that track the performance of different types of stocks. For example, the SPX index tracks the 500 largest companies listed on stock exchanges in the U.S.

The different asset classes are not detached from each other but instead show relationships with each other. For example, the increase in bond yields in the U.S. — as has been the case in 2021 — has led to a sell off in technology stocks (Nasdaq) as their growth has usually been financed through loans which now have become more expensive as interest rates rise or are feared to rise. This money obtained from selling technology equities has been invested in alternative asset classes such as commodities and even in emerging cryptocurrencies as well as in equities in different sectors.

In financial markets there are a variety of instruments used with the intention to earn premium from the investment. These instruments include derivatives that are tied to the underlying asset's price and allow investors to earn large returns from small investments. The derivatives consist of but are not limited to forward contracts, (call and put) options and swaps. For instance, a forward contract is a contract between two parties to buy or sell an asset for a predetermined price on a later date. An example of this would be Ryanair agreeing to buy a certain amount of fuel in the future for a price determined today. Such a strategy is hedging which one uses to protect itself from unfavourable price changes. Understanding the existence of the different derivatives and how they function is important because, for example, in the case

of Ryanair hedging its fuel price risk, hedging could result in greater losses or greater profits for Ryanair. This in turn would be reflected in the value of the equity of Ryanair, in effect the value of its share price.

Chapter 3

Tensorization Examples

In this chapter examples of the different tensorization methods explained in chapter 2, section 2.1.3, are implemented. The purpose of this is to illustrate the different tensorization methods with simple data so that the tensorization methods can be presented concisely before deciding on their applicability in a financial context. Two examples are walked through, namely, univariate time series and multivariate time series, although the tensorization methods could be extended for higher-order tensors as well with some exceptions.

3.1 Example 1 — Univariate Time Series

Consider the case of univariate time series of length L. Let the time series be stored in vector $\mathbf{p} \in \mathbb{R}^L$. Furthermore, assume that $L = 7$ and that the time series recorded was as follows:

$$\mathbf{p} = \begin{bmatrix} 2 & 4 & 6 & 8 & 10 & 12 & 14 \end{bmatrix}^T . \tag{3.1}$$

Since we are dealing with univariate time series in this example, multivariate tensorization techniques (MPR / vector-variate regression) as well as convolution tensor are not applicable.

3.1.1 Reshaping

Consider that the time series stored in vector $\mathbf{p}$ represents one public companys' closing share price of its stock. Assume that the first three entries in vector $\mathbf{p}$ represent the company's average closing share price of the stock from February to April and the rest four entries represent the company's average closing share price of the stock from May to August. Furthermore, assume that the company has not recorded its January's average closing share price of the stock. To resolve this, the vector is padded such that the company's record from February is duplicated for January. The new time series is then

$$\tilde{\mathbf{p}} = \begin{bmatrix} 2 & 2 & 4 & 6 & 8 & 10 & 12 & 14 \end{bmatrix}.$$

Now assume that the company has a practice of comparing its closing share price of the stock for every four months. Vector $\tilde{\mathbf{p}}$ can now be folded into a matrix where the number of rows corresponds to four quarters as desired and each entry in the matrix represents its closing share price of the stock for a particular month. This is follows:

$$\tilde{\mathbf{P}} = \begin{bmatrix} 2 & 8 \\ 2 & 10 \\ 4 & 12 \\ 6 & 14 \end{bmatrix}.$$

3.1.2 Hankelization

Applying Hankelization to the vector in (3.1) with $L = I + J - 1 = 7$ and choosing $I = 4$ so that $J = 4$. Thus, the Hankel matrix becomes $\mathbf{P} = \mathcal{H}_{I,J}(\mathbf{p})$ such that

$$\mathbf{P} = \mathcal{H}_{I,J}(\mathbf{p}) = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 4 & 6 & 8 & 10 \\ 6 & 8 & 10 & 12 \\ 8 & 10 & 12 & 14 \end{bmatrix}.$$

On the other hand, the Toeplitz matrix of the vector $\mathbf{p}$ would be

$$\mathbf{P} = \mathcal{T}_{I,J}(\mathbf{p}) = \begin{bmatrix} 8 & 10 & 12 & 14 \\ 6 & 8 & 10 & 12 \\ 4 & 6 & 8 & 10 \\ 2 & 4 & 6 & 8 \end{bmatrix}.$$

As such, the Toeplitz matrix is a permutation (or a vertically flipped version) of the Hankel matrix.

3.1.3 Löwner Folding

To apply Löwnerization to the vector $\mathbf{p}$, one needs to choose the evaluation points. Assume that the evaluation points are $\mathbf{t} = [1\ 2\ 3\ 4\ 5\ 6\ 7]$. Then, Löwnerization of the vector $\mathbf{p}$ yields

$$\mathbf{L}(\mathbf{p}) = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}.$$

It is clear that Löwner matrix is not suitable in financial time series forecasting as little information would be obtained from a matrix with constant entries as the example illustrates. This is supported by their use case which is often fitting a model to data through rational approximations [13]. As such, it will be excluded from this point onward.

3.2 Example 2 — Multivariate Time Series

Consider the case of multivariate time series of lengths $L_1 = L_2 = L$. These time series could represent, for example, a particular company's revenues and total costs over L days.

3.2.1 Reshaping

Assume that the different time series are stored in vector $\mathbf{p} \in \mathbb{R}^{L_1 L_2}$. This is obviously messy and a matrix representation would provide a clearer way of visualizing the different time series. As such, the matrix would become $(L_1 = L_2 = L) \times (2L_1/L_1 = 2L_2/L_2 = 2L/L = 2)$:

$$\mathbf{P} = \begin{bmatrix} p(1) & p(L_1+1) \\ p(2) & p(L_1+2) \\ \vdots & \vdots \\ p(L_1) & p(2L_1) \end{bmatrix}.$$

For example, if $\mathbf{p} = [20, 21, 22, 23, 25, 15, 15, 16, 17, 19]$, then

$$\mathbf{P} = \begin{bmatrix} 20 & 15 \\ 21 & 15 \\ 22 & 16 \\ 23 & 17 \\ 25 & 19 \end{bmatrix},$$

and it is easy to see that the firm has made profit over each of the five days.

3.2.2 Hankelization

Consider that the two time series (same as defined in section 3.2.1) are represented by a matrix $\mathbf{P}$ (for example the resulting matrix from reshaping in section 3.2.1) as follows:

$$\mathbf{P} = \begin{bmatrix} 20 & 15 \\ 21 & 15 \\ 22 & 16 \\ 23 & 17 \\ 25 & 19 \end{bmatrix}.$$

In the matrix the temporal mode is represented by the rows and the features corresponding to revenues and total costs are represented by the columns. The matrix $\mathbf{P}$ can now be tensorized to arrive at a higher-order tensor. In [8] to arrive at a Hankel tensor from an input tensor, Hankelization was applied on the temporal mode only. This is because the relationship between neighbour items of multiple time series is often not stronger than their temporal correlation [8]. Applying Hankelization along the temporal mode of the matrix $\mathbf{P}$ yields a third-order tensor $\mathcal{P} \in \mathbb{R}(3 \times 3 \times 2)$ where the different Hankel matrices are stacked along the third mode. Slices of $\mathcal{P}$ are as follows:

$$\mathcal{P}_{::1} = \begin{bmatrix} 20 & 21 & 22 \\ 21 & 22 & 23 \\ 22 & 23 & 25 \end{bmatrix}, \ \mathcal{P}_{::2} = \begin{bmatrix} 15 & 15 & 16 \\ 15 & 16 & 17 \\ 16 & 17 & 19 \end{bmatrix}.$$

3.2.3 Convolution Tensor

As shown in section 2.1.3, a convolution tensor $\mathcal{C}$ can be used to transform vectors into Toeplitz matrices when the vectors are convolved with each other. However, there was little application/literature found in directly convolving differently financial time series with each other. As such, convolution tensor will be discarded in the rest of this project as its use cases do not align with the goals of this project. On the other hand, convolution layers are often used in NNs to improve the network's performance. Therefore, convolution tensors can naturally be used in ML/DL applications even in a financial context.

3.2.4 Tensor Structures in Vector-variate Regression

Consider that the two time series are stored in vectors $\mathbf{p}_1$ and $\mathbf{p}_2$ and they are used to estimate the firm's profits for which no records exist in this example. Assume that the profits can be linearly approximated, that is

$$y = h(\mathbf{p}_1, \mathbf{p}_2) = \begin{bmatrix} 1 & \mathbf{p}_1^T \end{bmatrix} \begin{bmatrix} w_0 & \mathbf{w}_2^T \\ \mathbf{w}_1^T & \mathbf{W}_{12} \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{p}_2 \end{bmatrix}.$$

If estimating the firm's profits cannot be estimated through the simple model above but requires more complex functional behaviours, then the model can include more terms. This can be achieved, for example, through a quadratic polynomial regression as follows:

$$y = h(\mathbf{p}_1, \mathbf{p}_2) = \begin{bmatrix} 1 & \mathbf{p}_1^T & (\mathbf{p}_1 \otimes \mathbf{p}_1)^T \end{bmatrix} \mathbf{W} \begin{bmatrix} 1 \\ \mathbf{p}_2 \\ \mathbf{p}_2 \otimes \mathbf{p}_2 \end{bmatrix},$$

where the weights matrix $\mathbf{W}$ is defined as

$$\mathbf{W} = \begin{bmatrix} w_0 & \mathbf{w}_2^T & \text{vec}(\mathbf{W}_{22})^T \\ \mathbf{w}_1 & \mathbf{W}_{12} & [\mathcal{W}_{122}]_{(1)} \\ \text{vec}(\mathbf{W}_{11}) & [\mathcal{W}_{112}]_{(1,2)} & [\mathcal{W}_{1122}]_{(1,2)} \end{bmatrix}.$$

The above examples illustrate the tensor structures in vector-variate regression. However, in this project the aim is to use NNs in financial time series forecasting instead of simply building a vector-variate regression model. Therefore, even though vector-variate regression has the inherent tensor structure, it will not be considered in the rest of this paper as the method does not align with the goals of this project. However, the method could be utilized in other regression problems even in a financial context.

3.3 Summary

This chapter presented examples of tensorizing univariate time series and multivariate time series using the different tensorization methods explained in chapter 2, section 2.1.3. The purpose was to further explore their suitability in time series forecasting with a more practical approach other than relying purely on theory. After implementing the different tensorization methods on the examples presented, it was concluded that reshaping and Hankelization show the most promise out of the different tensorization methods explained in financial time series forecasting using NNs. However, as illustrated, reshaping is a rather simple tensorization

method and often used indirectly in various tensorization algorithms and operations as will be seen in chapter 5, section 5.3. Therefore, only Hankelization is considered from this point onward in the empirical work. The rest of the tensorization methods have their own applications but they are not necessarily that well suited in a financial context.

Chapter 4

Financial Data

In this chapter financial data used in the empirical work is presented. In doing so, the choice of data is explained after which pre-processing, stationarity and memory are discussed.

4.1 Choice of Data

In this project the aim is to forecast returns for the next day of the SPX index. However, the code implemented can straightforwardly be modified to forecast the returns of other assets as well. It is natural to include historical data of the SPX when forecasting its future returns. However, historical data of the SPX should not be taken standalone when predicting its future return. This is because assets over various asset classes show relationships with each other as discussed in chapter 2, section 2.3. For example, generally stocks and gold are inversely related meaning that when the price of gold goes up, valuations of stocks go down. Moreover, bond yields tend to affect valuations of stocks, in particular technology stocks. Therefore, it was chosen to include several assets across equities, fixed income, cash equivalents and commodities. Table 4.1 shows the assets included in the forecasting problem.

The U.S., German and Japanese stock market indices were chosen as they reflect the state of the economies in America, Europe and Asia. Similar reasoning was used in selecting fixed income

Table 4.1: Assets included in forecasting the next day returns of SPX.

Asset class	Asset	Ticker symbol
Equities	U.S. stock market index (S&P 500)	SPX
Equities	German stock market index	DAX
Equities	Japanese stock market index (Nikkei 225)	NKY
Fixed income	U.S. 10-year bond yield	US10YT
Fixed income	Germany 10-year bond yield	DE10YT
Fixed income	Japan 10-year bond yield	JP10YT
Cash equivalents	Pound sterling	USDGBP
Cash equivalents	Euro	USDEUR
Cash equivalents	Japanese yen	USDJPY
Commodities	Gold futures	ZG
Commodities	Crude oil WTI futures	T
Commodities	Natural gas futures	NG

and cash equivalents assets. Gold, crude oil and natural gas were chosen as the commodities as they are included in the most actively traded commodities [44]. The choice of assets needs to be done with care to mitigate the criticism of ML models being labeled as black box models. It is infeasible to include a very large number of arbitrarily chosen assets to the models as the complexity increases as well as the computations needed to be made and the training time. Therefore, relying on theory or any a priori ideas when choosing the assets is important [45]. This helps to make the models more interpretable as well as the forecasts more reliable. Figure 4.1 shows a visualization of the fourth-order financial tensor.

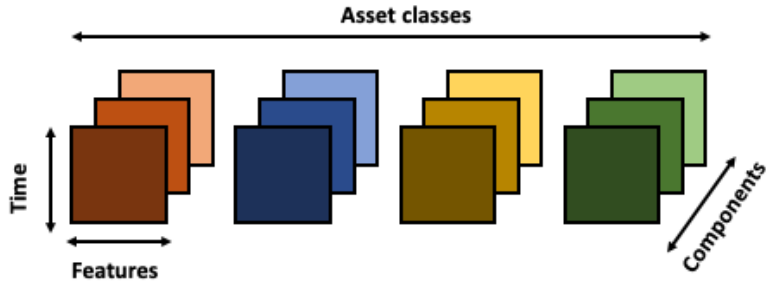


Figure 4.1: Multi-modal tensor representation of the financial data

4.2 Pre-Processing of Data

All of the data of the assets shown in table 4.1 was retrieved from Investing.com [46]. A benefit of retrieving all of the data from the same source is that they are uniformly formatted which makes pre-processing of the data more straightforward. The data range was chosen to be from February 2006 to April 2021. All of the assets had their closing, opening, high and low values recorded. Some of the assets had their volume records as well but often they were missing. Therefore, it was decided not to include volume as one of the features. The number of features was a limitation of using Investing.com as the data source. This is because some other data sources had more features available for the assets. However, those alternative data sources often required a paid subscription or did not allow the user to download the data.

Since public holidays, for example, affect the opening days of stock markets, not all of the assets had the same number of records available. To solve this problem while minimizing the number of artificial records introduced to the data, the number of records in the different time series were reduced to have the same number of records as NKY which had the shortest time series. Then, if the various time series still had blank entries for some particular days, the next day's records were duplicated such that each day now has a record associated with it. Having done the above, the data (for now) is $\mathcal{X} \in \mathbb{R}^{3757\times4\times3\times4}$, where the modes of $\mathcal{X}$ are time, features, components and asset classes. Figures 4.2—4.5 show plots of the different assets to verify that the data looks as intended.

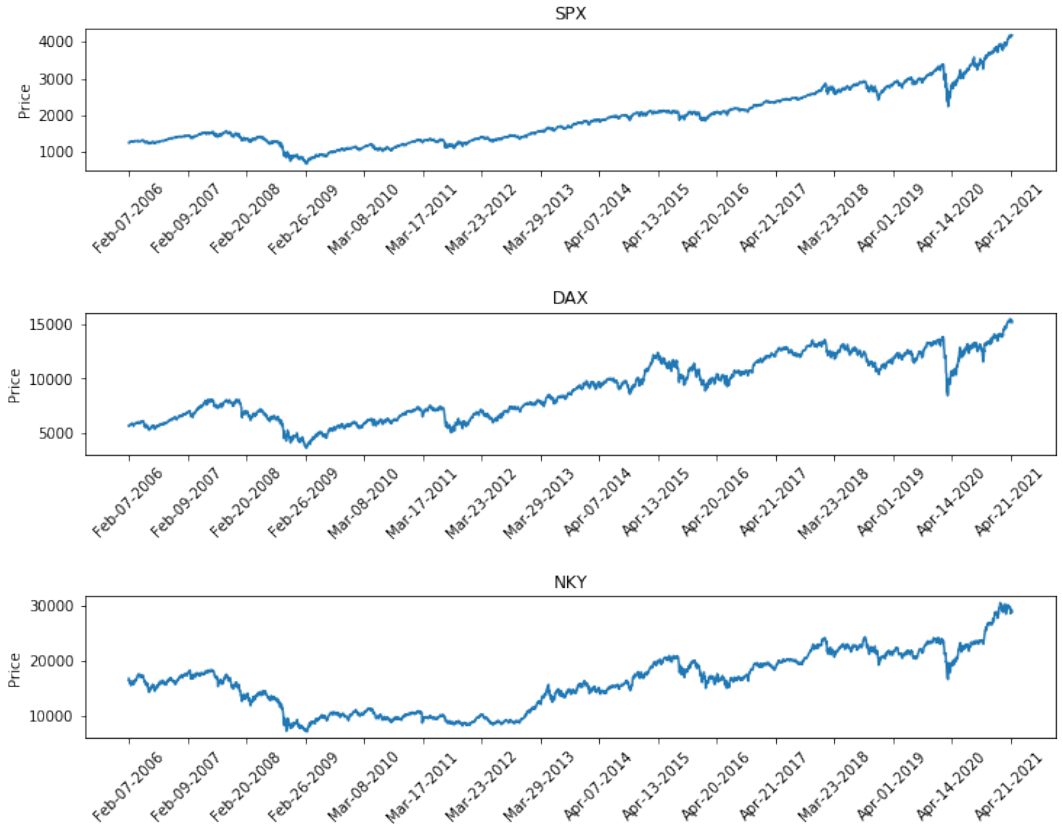


Figure 4.2: Stock market indices' prices (asset class: equities)

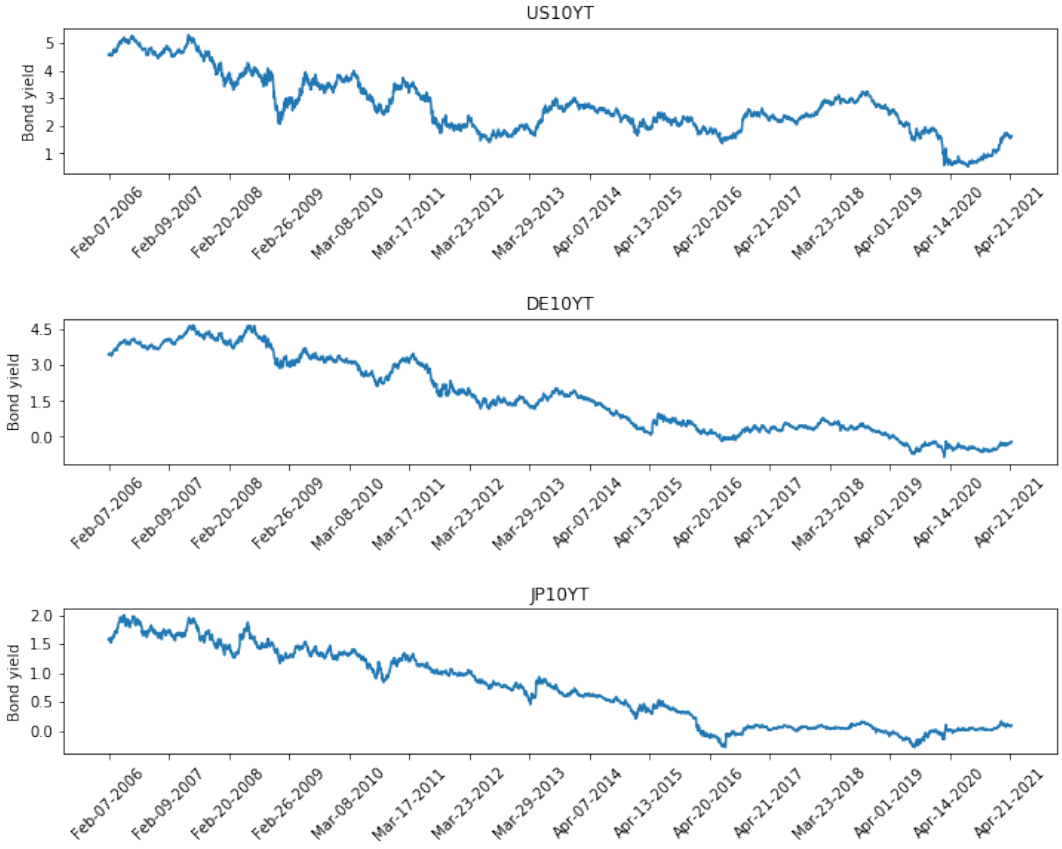


Figure 4.3: Countries' bond yields (asset class: fixed income)

Figure 4.4: Currencies' exchange rates (asset class: cash equivalents)

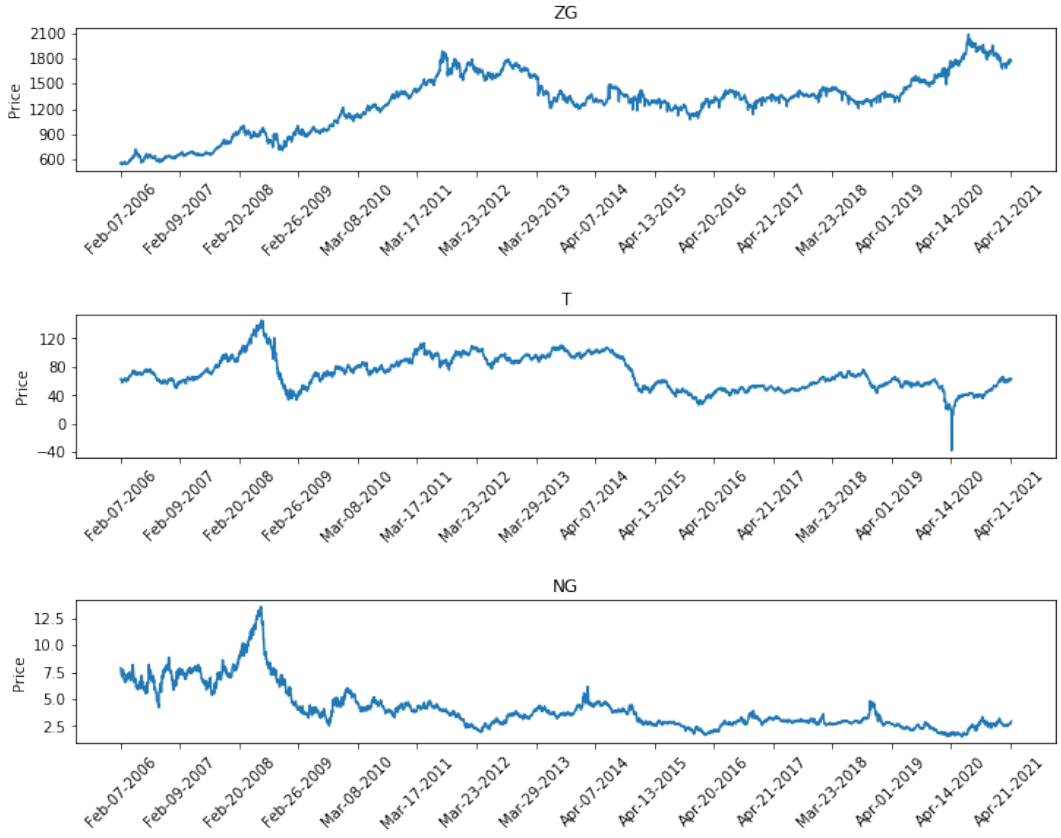


Figure 4.5: Futures' prices (asset class: commodities)

4.3 Stationarity of Data

The concept of stationarity was briefly touched upon in chapter 2, section 2.2.2. If stochastic signal's unconditional joint probability distribution does not change when shifted in time, the signal is considered stationary. This means that its statistical properties such as mean and variance stay constant over time. A weaker form of stationarity, wide-sense stationarity (WSS), is often used in signal processing. WSS for a signal means that its first moment (mean) needs to be time invariant and its autocorrelation function depends on time differences only. For signal X these properties are defined as

$$m_X(t) = m_X(t+T),\ \forall t \in \mathbb{R}, \tag{4.1}$$

and

$$R_{xx}(t_1, t_2) = R_{xx}(t_1 - t_2),\ \forall t_1, t_2 \in \mathbb{R}. \tag{4.2}$$

In financial applications stationary signals are important because this means that one can create more accurate forecasts of the signals to predict, for example, price movements. Furthermore, non-stationary signals are unpredictable and thus their modeling and forecasting are difficult. [47] In ML application this means that the trained model will have worse performance. Generally speaking, in supervised learning if the features are non-stationary, one cannot map the new observation to a large number of known examples. Supervised learning problems typically require stationary features as stationarity is a necessary, non-sufficient condition for the good performance of a ML algorithm. [48]

From figure 4.2 it can be seen that the time series of the different stock market indices tend to increase over time. Such a behaviour is naturally good for the economy but not so good for modelling because the time series show non-stationary behaviour. To transform the price time series stationary, differencing can be used. Differencing can be achieved by first log-transforming the price data and then differencing consecutive samples. This yields log returns

which is defined as follows [49]:

$$r_t^{\log} = \ln\left(\frac{p_t}{p_{t-1}}\right) = \ln\left(p_t\right) - \ln\left(p_{t-1}\right), \tag{4.3}$$

where p_t is the given asset's price at time t.

Figure 4.6 shows histograms of closing prices and log returns computed using closing prices of the SPX. The figure shows that the histogram of the log returns resembles normal distribution whereas the prices do not show Gaussian behaviour. This is notable since prices are often assumed to be log-normally distributed in the short run in quantitative finance which implies that the log returns will be normally distributed [47]. Furthermore, the data showing Gaussian properties will help with training a good ML model since ML algorithms often assume Gaussian distribution, and thus the log returns being concentrated around zero is convenient [49].

However, there is a trade-off between stationarity and memory. Memory refers to a long history of previous levels that shift the time series' mean over time. By differencing the time series, some memory is always erased. [48] This can be mitigated by introducing new stationary features to the data. Such an approach is used in this project and discussed next.

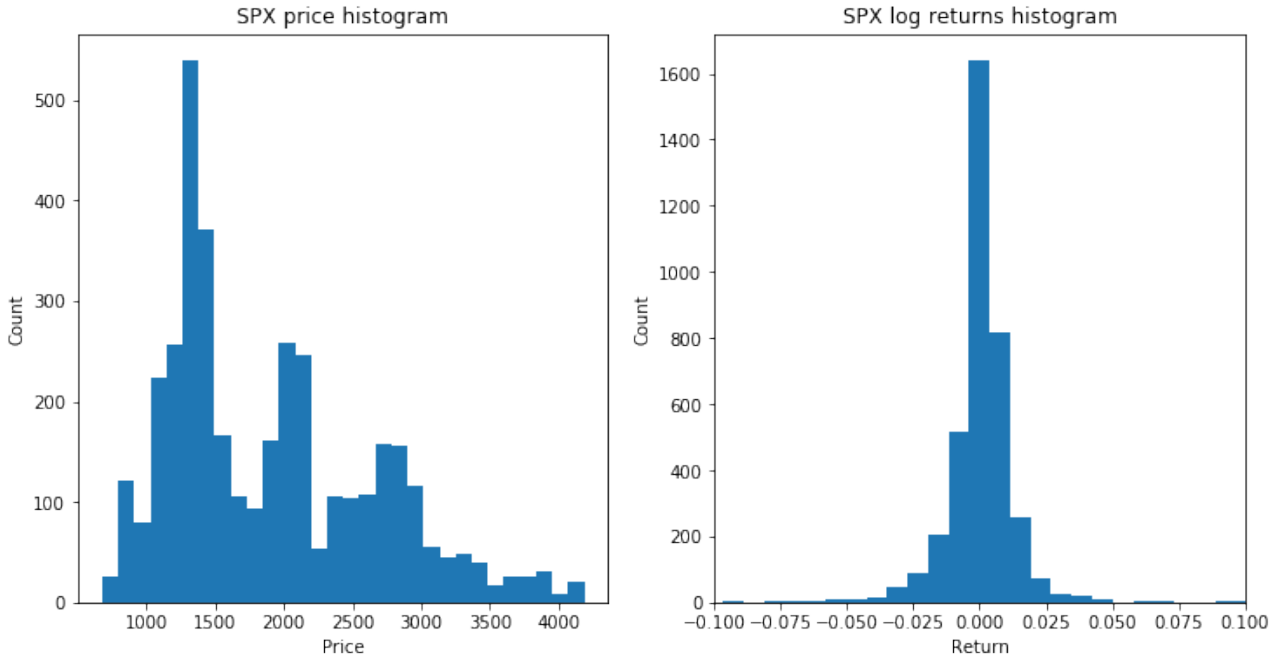


Figure 4.6: Histograms of closing prices and log returns computed using closing prices of SPX.

4.4 Memory Adding Features

The previous section (section 4.3) explained benefits of using log returns instead of prices. Therefore, log returns is added as a new feature to tackle the stationarity issue. However, the use of log returns erases some memory which can be mitigated by introducing additional memory through new features. As such, the following features are introduced: log returns, log returns moving average, log returns moving average standard deviation, log returns moving average skew, log returns moving average kurtosis, high-low spread (HLS) and relative high-low-close (RHLC) [23]. Skewness measures the distribution's symmetry or lack of symmetry whereas kurtosis measures the distribution's tailedness relative to normal distribution. For the features involving moving averages, these are computed on three separate occasions with window sizes of 5, 10 and 22 days. These window sizes correspond to the number of trading days per week, two weeks and month. HLS and RHLC are respectively computed as follows [23]:

$$HLS = \frac{P_H(t) - P_L(t)}{P_L(t)}, \tag{4.4}$$

$$RHLC = \frac{P_C(t) - P_L(t)}{P_H(t) - P_L(t)}, \tag{4.5}$$

where $P_H(t), P_L(t)$ and $P_C(t)$ refer to high, low and closing prices for day t, respectively.

Having introduced the new features explained above, the data now has $4 \times 3 + 3 = 15$ features in total. The features along with their short descriptions are summarized in table 4.2. The data tensor is a fourth-order tensor $\mathcal{X} \in \mathbb{R}^{3735\times15\times3\times4}$, where the modes of $\mathcal{X}$ are time, features, components and asset classes. Each time step, t, can be described through a third-order tensor, $\mathcal{Z}_t \in \mathbb{R}^{15\times3\times4}$.

Figures 4.7 and 4.8 show histograms of the newly introduced features of the SPX. From the figures it can be seen that most of the new features show Gaussian behaviour.

Table 4.2: Data features summarized

Feature	Description
LR	Log returns computed using closing prices
LRMA5	Moving average of log returns using a window size of 5 days
LRMA10	Moving average of log returns using a window size of 10 days
LRMA22	Moving average of log returns using a window size of 22 days
LRSTDMA5	Moving average of log returns' standard deviation using a window size of 5 days
LRSTDMA10	Moving average of log returns' standard deviation using a window size of 10 days
LRSTDMA22	Moving average of log returns' standard deviation using a window size of 22 days
LRSMA5	Moving average of log returns' skew using a window size of 5 days
LRSMA10	Moving average of log returns' skew using a window size of 10 days
LRSMA22	Moving average of log returns' skew using a window size of 22 days
LRKMA5	Moving average of log returns' kurtosis using a window size of 5 days
LRKMA10	Moving average of log returns' kurtosis using a window size of 10 days
LRKMA22	Moving average of log returns' kurtosis using a window size of 22 days
HLS	High-low spread
RHLC	Relative high-low-close

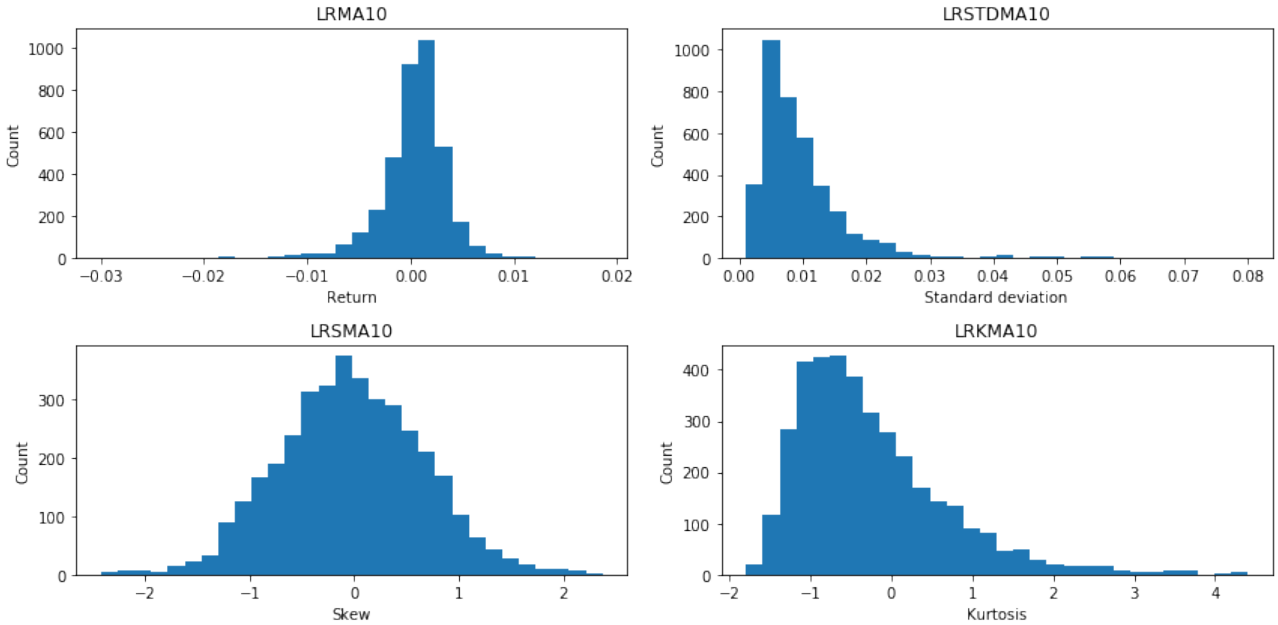


Figure 4.7: Histograms of features involving moving average (window size has been set to 10) of the SPX

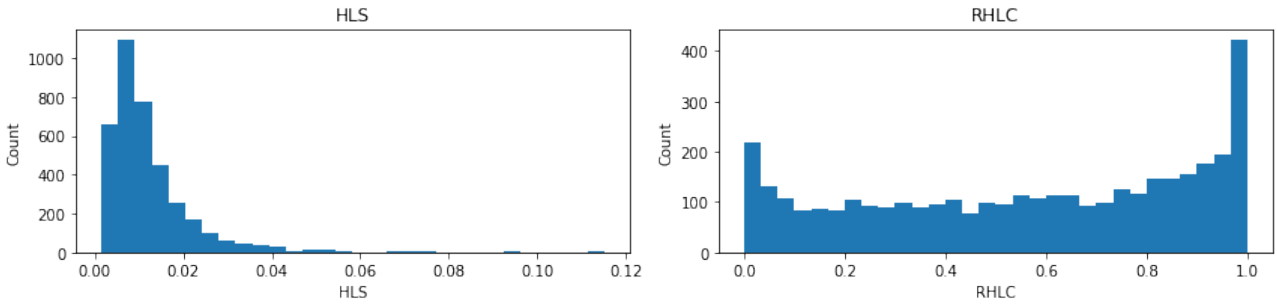


Figure 4.8: Histograms of HLS and RHLC of the SPX

4.5 Labelling of Data

Supervised learning problems need labels which represent the ground truth that one would want the model to predict. In this project the labels are represented by the SPX log returns time series which are shifted one time step back. The reason for shifting the time series is to make the models learn to forecast the next day returns at time $t+1$ using past features at time t. Furthermore, the problem of forecasting the next day log returns of SPX is characterized as a classification problem in which the labels $\{0, 1, 2\}$ represent negative, neutral and positive returns, respectively. Finally, the labels are one-hot encoded as $\{100, 010, 001\}$. [50], [23]

4.6 Final Pre-processing Touches

A split of 90/10 was applied to the data such that 90% of the available data is used for training and the rest 10% for testing. Then, a further split of 90/10 was applied to the training set such that 10% of the training data was used for validation purposes and the rest for training purposes. Moreover, the data was further pre-processed by applying scaling (standardization) to it so that each of the features has zero mean and variance of one. Scaling is important because ML algorithms tend to weigh greater values as higher values and smaller values as lower values. This happens regardless of the data unit. [51] Furthermore, if the features have a similar scale, this helps the gradient descent converge more smoothly and quickly to the minimum [52].

Chapter 5

Methodology

In this chapter implementations of the NN models are explained. In doing so, a standard RNN, tensor-train RNN (TT-RNN) as well as Hankel tensor tensor-train RNN (H-TT-RNN) models are covered. Additionally, model performance evaluation metrics are discussed.

All of the models were implemented in Python 3 using Keras. Keras is a deep learning framework running on top of Theano or TensorFlow. It is useful as it allows the user to easily and efficiently implement complex NNs.

5.1 Recurrent Neural Network Model

It is important to note that the RNN model requires some further pre-processing of the data to work properly. In essence, samples of the data need to be matricized such that mode-1 and mode-2 are time steps and features, respectively. Different time steps were considered such as 5, 10 and 22 which correspond to trading days per week, two weeks and month, respectively. Before elaborating more about the results, it was found that using a time step of 22 achieved the best results. The samples were then obtained from the data using a sliding window having a size of the time step. This means that each sample is represented by $\mathbf{X}_{\text{sample}} \in \mathbb{R}^{22\times 180}$ and obtained from the tensor $\mathcal{X} \in \mathbb{R}^{T\times 22\times 180}$ ($180 = 15 \times 3 \times 4$).

The RNN model implemented consists of two layers, recurrent layer and fully connected layer. The number of units for the recurrent layer was limited to 1024 as the computation time increased unbearingly for larger number of units when tested. The fully connected layer consisted of 3 units. Recurrent layer and fully connected layer used `tanh` and `softmax` activation functions, respectively. Additionally, the learning rate was set to 10^{-3} which is often used in ML problems and the batch size was set to be 66 (this batch size corresponds to the number of trading days per one quarter, in effect, 22×3).

5.2 Tensor-Train Recurrent Neural Network Model

The hidden state of a traditional RNN was defined in (2.35). This can be modified to be applicable in TT-format as follows:

$$\mathbf{h}_t = \phi_h(W_h\mathbf{h}_{t-1} + \mathrm{TT}(W_x\mathbf{x}_t) + \mathbf{b}_h). \tag{5.1}$$

This can be interpreted such that the folding of $\mathbf{h}_t \in \mathbb{R}, \mathbf{x}_t \in \mathbb{R}^P$, and $\mathbf{b} \in \mathbb{R}^M$ into respective order-N tensors, $\mathcal{H}_t \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}, \mathcal{X}_t \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, and $\mathcal{B} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$ allows one to express $\mathbf{W}_x \in \mathbb{R}^{M \times N}$ in TT-format as in (2.32) [3], [23]. Thus, whereas $\mathbf{W}_x$ had exponential parameter complexity $\prod_{n=1}^{N} I_n J_n = PM$, the complexity is compressed by the TT-RNN to linear $\sum_{n=1}^{N} I_n J_n R_{n-1} R_n$ in the dimensions I_n and J_n [23]. This is particularly efficient for small mode sizes and TT-ranks and shows improvements in the storage complexity, computational complexity and training time [3], [23].

In chapter 4, section 4.4 it was mentioned how the tensor for each time step, t, is a third-order tensor $\mathcal{Z}_t \in \mathbb{R}^{15 \times 3 \times 4}$. For the TT-RNN model, the tensor $\mathcal{Z}_t$ was further reshaped into a fifth-order tensor, $\mathcal{X}_t \in \mathbb{R}^{1 \times 5 \times 3 \times 4 \times 3}$. Note that the modes of the tensor $\mathcal{X}_t$ were chosen such that the first three modes correspond to the 15 features and the last two correspond to the asset classes and components, respectively [23]. The use of fifth-order tensor improves the compression because of smaller mode sizes and results without loss in physical interpretability [23]. This is because the modes representing features, components and asset classes are not

mixed. [23] Before elaborating more about the results, different variations of $\mathcal{X}_t$ were tested such as $\mathcal{X}_t \in \mathbb{R}^{1\times3\times5\times4\times3}$ but these had worse performances.

Similar to the RNN model, the neural network consisted of one recurrent layer having 1024 units and one fully connected layer having 3 units. The recurrent and fully connected layers used `tanh` and `softmax` activation functions, respectively. The learning rate was also set to 10^{-3} and the time step and batch size were set to be 22 and 66, respectively. Finally, different TT-ranks of R_1, R_2, R_3 and R_4 for the model were empirically tested. It was found that using TT-ranks of $R_i = 3$ for $i = 1, \ldots, 4$ achieved the best results results and that using lower and higher values resulted in under-fitting and over-fitting, respectively. This implies that the TT-ranks could be used not only for compression but also for regularization [23].

5.3 Hankel Tensor Tensor-Train Recurrent Neural Network Model

Hankelizing a time series as a concept is relatively straightforward as illustrated by the examples in chapter 3. Hankelization allows one to arrive at higher-order tensors given lower-order input tensors and provides an extension for TT-RNN models discussed in the previous section. Multi-way delay embedding transform (MDT) is a technique to Hankelize input data [7]. For example, a standard delay embedding transform of a time series stored in vector, $\mathbf{v} = (v_1, v_2, \ldots, v_L)^T \in \mathbb{R}^L$ with τ is defined as [7]

$$\mathcal{H}_\tau(\mathbf{v}) = \begin{bmatrix} v_1 & v_2 & \cdots & v_{L-\tau+1} \\ v_2 & v_3 & \cdots & v_{L-\tau+2} \\ \vdots & \vdots & \ddots & \vdots \\ v_\tau & v_{\tau+1} & \cdots & v_L \end{bmatrix} \in \mathbb{R}^{\tau\times(L-\tau+1)}. \tag{5.2}$$

Thus, it is evident from (5.2) that applying a standard delay embedding transform to a vector produces a Hankel matrix. Furthermore, if $\mathbf{S} \in \{0,1\}^{\tau(L-\tau+1)\times L}$ is a duplication matrix

satisfying [7]

$$\text{vec}(\mathcal{H}_\tau(\mathbf{v})) = \mathbf{S}\mathbf{v}, \tag{5.3}$$

then the standard delay embedding transform is given by [7]

$$\mathcal{H}_\tau(\mathbf{v}) = \text{fold}_{(L,\tau)}(\mathbf{S}\mathbf{v}). \tag{5.4}$$

Figure 5.1 shows an illustration of the delay embedding transform applied to a vector.

MDT can easily be applied to multiple time series. The financial data used in this project can be stored in a second-order tensor, $\mathbf{X} \in \mathbb{R}^{I \times T}$, where I refers to the various time series (in this case $I = 15 \times 3 \times 4 = 180$) and T is the number of time steps. Thus, applying MDT for $\mathbf{X}$ using duplication matrix $\mathbf{S} \in \mathbb{R}^{T \times \tau(T-\tau+1)}$ with τ along the temporal direction only because the relationship between neighbour items of multiple time series is often weaker than their temporal correlation, generates a third-order block Hankel tensor $\hat{\mathcal{X}} \in \mathbb{R}^{I \times \tau \times (T-\tau+1)}$ [8]. Moreover, since the input tensor is a matrix, then $\mathbf{X} \times_N \mathbf{S} = \mathbf{X}\mathbf{S}^T$. Note that the block Hankel tensor is assumed to be low-rank in the embedded space [8], [7]. Figure 5.2 shows an illustration of applying MDT along the temporal mode of a second-order tensor consisting of several time series.

To arrive at the H-TT-RNN model, MDT was applied along the temporal mode of the input tensor as suggested by [8]. The choice of τ has an impact on the size of the input tensor and hence on the complexity through the training time. Different values for τ were tested and it was found that $\tau = 3$ achieved good results. In [8] it was concluded the the hyper-parameter

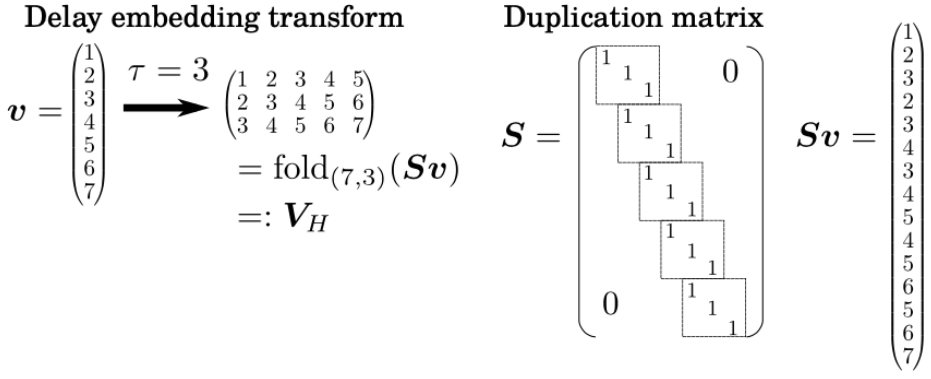


Figure 5.1: Delay embedding transform applied to a vector. Source: adapted from [7].

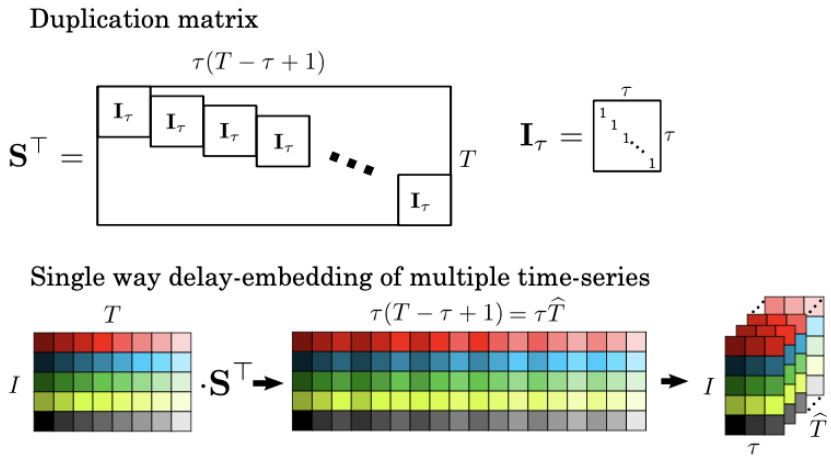


Figure 5.2: Applying MDT along the temporal mode of a a second-order tensor consisting of multiple time series. Source: adapted from [8].

tuning with respect to τ is not that important but smaller values of τ such as $\tau = 2-4$ achieved better results than large values of τ.

After applying MDT to the matrix, $\mathbf{X} \in \mathbb{R}^{I \times T}$, the shape of the new tensor, $\hat{\mathcal{X}} \in \mathbb{R}^{I \times \tau \times (T-\tau+1)}$, is invalid for the RNN. To resolve this issue, the tensor was matricized back into a matrix, $\hat{\mathbf{X}} \in \mathbb{R}^{\tau(T-\tau+1) \times I}$. This reshaping operation can however be avoided if one simply multiplies the input matrix, $\mathbf{X}$, with the duplication matrix, $\mathbf{S}^T$, effectively computes $\mathbf{X}\mathbf{S}^T$, and does not proceed to fold the resulting matrix into a higher-order tensor, $\hat{\mathcal{X}}$. However, if one instead considered a TT fully connected NN, thus discarding its intrinsic limitation with regards to temporal dependencies in the data, then the Hankelized tensor, $\hat{\mathcal{X}}$, could be fed as an input to the network in its natural form. If this is not desired, another option would be to modify the RNN and apply TT decomposition to the input [50], [19]. Such a method is however beyond the scope of this project.

Note that each time step of the matrix, $\hat{\mathbf{X}}$, is actually represented by a third-order tensor, $\hat{\mathcal{Z}}_{\hat{t}} \in \mathbb{R}^{15 \times 3 \times 4}$, where the modes correspond to features, components and asset classes, respectively. Analogous to the TT-RNN model, $\hat{\mathcal{Z}}_{\hat{t}}$ was further reshaped into a higher-order tensor, $\hat{\mathcal{D}}_{\hat{t}} \in \mathbb{R}^{1 \times 3 \times 5 \times 4 \times 3}$, where the first three modes correspond to the number of features and the last two correspond to the asset classes and components, respectively. Finally, different TT-ranks of $R_1, R_2, R3$ and R_4 for the model were empirically tested. Using TT-ranks of $R_i = 1$ for

$i = 1, \ldots, 4$ achieved the best results.

Similar to the RNN and TT-RNN models, the H-TT-NN model consisted of one recurrent layer having 1024 units and one fully connected layer having 3 units that used `tanh` and `softmax` activation functions respectively. This decision was made in order to allow for a better comparison between the models. Furthermore, the time step and batch size were set to be $(22 \times \tau - 1)$ and $(66 \times \tau - 1)$, respectively.

5.4 Model Performance Evaluation Metrics

All of the models are evaluated according to same metrics to allow for a better comparison. First, the complexity of the models can be evaluated based on the number of parameters they use. For example, if two models have the same accuracy but the other model uses considerably more parameters, the model with fewer parameters is more desirable. Second, the models' performances are evaluated using the cross-entropy loss function as well as the accuracy of the forecast. Finally, some of the trading performance metrics used in [50] and [23] are applied in this project as well. These trading performance metrics are summarized in table 5.1 along with their short descriptions and preferences of the metric's value being high or low.

The metrics are rather self-explanatory apart from Sharpe ratio. Sharpe ratio helps investors to evaluate investment's return relative to its risk. Essentially, it is the average return earned in excess of the risk-free rate (for example government bond yield) per unit of volatility (or total risk), where volatility measures the price fluctuations of an asset or portfolio. [53]

Table 5.1: Trading performance metrics

Metric	**Description**	**Preference**
Total return (%)	Total log return	High
Average return (%)	Average return of all trades	High
Maximum drawdown (%)	Maximum loss from peak to trough	low
Maximum drawdown duration	Longest time between peaks	low
Sharpe ratio	Ratio of average returns to STD	High
Skew	Asymmetry of return distribution	High
Kurtosis	Tailedness of return distribution	Low

Chapter 6

Results

In this chapter results from the empirical part are presented, discussed and compared.

6.1 Recurrent Neural Network Model Results

6.1.1 Complexity

The complexity measure, in effect, the number of parameters showed that the RNN model used 1236995 parameters in total.

6.1.2 Training and Testing

Table 6.1 summarizes the losses and accuracies for the training and testing of the RNN model. The test accuracy was found to be 23.579545% which is poor and due to the over-fitting of the model. Figure 6.1 shows the cross-entropy losses and accuracies for both the training and validation. From the figure it can be seen that that the loss curves diverge from each other which is also true for the accuracy curves. This implies that the model is able to capture the temporal dependencies in the data while training but is over-fitting. Therefore, it is not able to generalise that well for the unseen data.

Table 6.1: RNN model's loss and accuracy results

Metric	Value
Train loss	0.807256
Validation loss	1.163723
Test Loss	1.250395
Train accuracy (%)	61.863559
Validation accuracy (%)	34.131736
Test accuracy (%)	23.579545

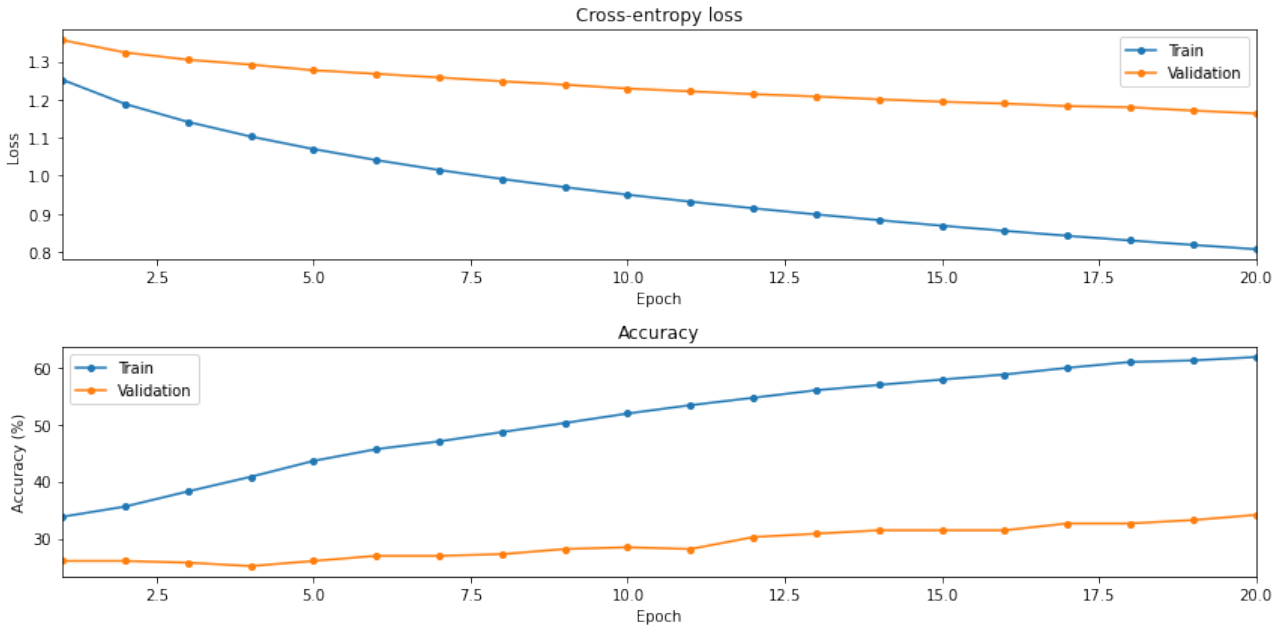


Figure 6.1: RNN model's cross-entropy loss and accuracy for training and validation datasets

6.1.3 Trading

Table 6.2 shows the values of the various trading performance metrics. As it can be seen from the table, the trading results of the RNN model are poor, implied by the large negative total return. However, this is expected due to over-fitting of the model. Figure 6.2 illustrates the actual long-only and trading returns by the model of the SPX. It can be seen from the figure that the RNN model's trading returns are worse than the actual cumulative returns.

Table 6.2: RNN model's trading results

Metric	Value
Total return (%)	-12.887654
Average return (%)	-0.036613
Maximum drawdown (%)	28.793144
Maximum drawdown duration	81
Sharpe ratio	-0.038219
Skew	-0.673402
Kurtosis	16.117341

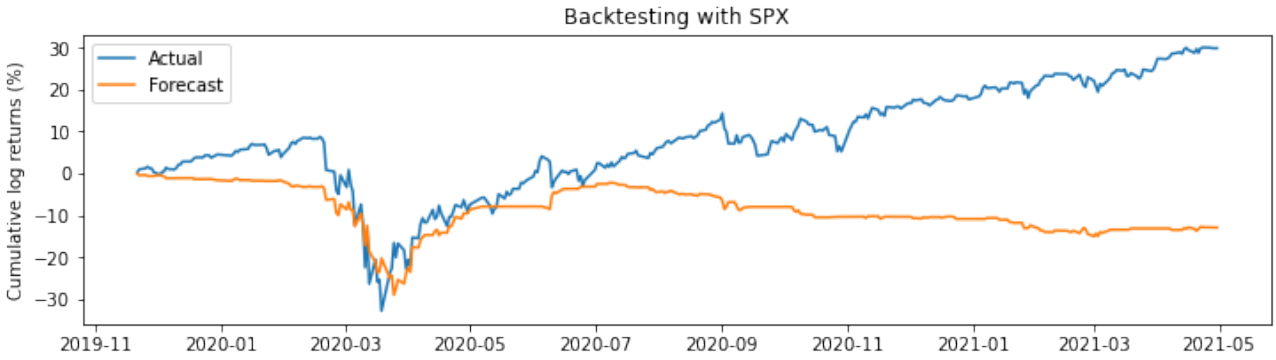


Figure 6.2: RNN model's backtesting with the SPX

6.2 Tensor-Train Recurrent Neural Network Model Results

Before focusing on the best performing model, table 6.3 displays the trading results for models of varying TT-ranks. From the table it can be seen that the total returns are worse for TT-ranks greater than 3 than they are for TT-ranks less than or equal to 3. The reason for this is that when the TT-ranks are large, less compression takes place meaning less regularization. As such, the models with large TT-ranks are over-fitting. It can be observed from the table that for TT-ranks of 3, the model has the best performance implied by the largest total return. Therefore, the rest of this section will focus on the model with TT-ranks of 3.

Table 6.3: TT-RNN model's trading results for varying TT-ranks. The best results are bolded.

Metric	TT-ranks									
	1	2	3	4	5	6	7	8	9	10
Total return (%)	9.712072	6.368348	**24.770336**	-18.543212	4.593882	-15.286434	-40.142120	4.791105	5.159584	0.960755
Average return (%)	0.027591	0.018092	**0.070370**	-0.052680	0.013051	-0.043427	-0.114040	0.013611	0.014658	0.002729
Maximum drawdown (%)	17.131367	21.384362	**10.842586**	29.273461	15.406494	21.948732	50.419025	17.597923	15.308145	20.402533
Maximum drawdown duration	**28**	67	38	258	160	125	154	39	110	266
Sharpe ratio	0.027145	0.016699	**0.072245**	-0.045988	0.012054	-0.041497	-0.107615	0.012592	0.015107	0.002632
Skew	-1.307563	-1.027037	1.198410	-1.428025	1.121405	0.697794	-1.892876	-0.844030	0.765457	**1.596376**
Kurtosis	**9.757519**	14.363780	10.044279	17.845567	16.003072	12.649394	12.393684	12.440362	9.975437	12.483128

6.2.1 Complexity

It is evident that the larger the ranks, the less memory efficient the model is. This is shown in figure 6.3. In the case of the standard RNN, the number of parameters used by the weight matrix $\mathbf{W}_x$ was $\prod_{n=1}^{N} I_n J_n = PM = 180 \times 4^5 = 184320$. However, the number of parameters of $\mathbf{W}_x$ was compressed to just $\sum_{n=1}^{N} I_n J_n R_{n-1} R_n = 480$ by TT decomposition which is 384 times fewer parameters. In total the number of parameters used by the TT-RNN model was 1053155.

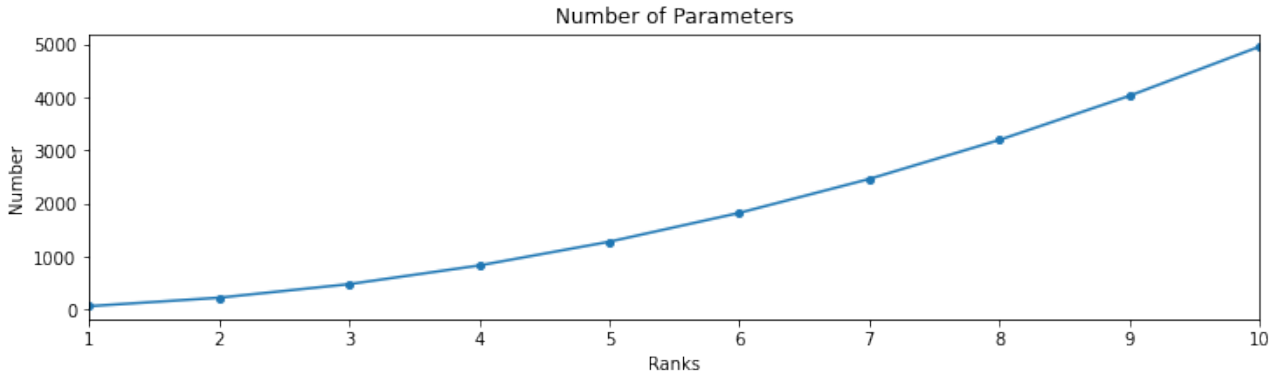


Figure 6.3: Number of parameters for different TT-ranks

6.2.2 Training and Testing

Table 6.4 summarizes the losses and accuracies of the TT-RNN model. The test accuracy of 51.704545% saw a major improvement when compared with the RNN model (23.579545%). Figure 6.4 shows the loss and accuracy for the train and validation curves of the TT-RNN model. From the figure it can be seen that the loss curves show a decreasing trend and that the accuracy curves either increase or stay approximately the same over the epochs.

Table 6.4: TT-RNN model's loss and accuracy results

Metric	Value
Train loss	0.791837
Validation loss	0.852382
Test loss	0.887323
Train accuracy (%)	57.038269
Validation accuracy (%)	50.898202
Test accuracy (%)	51.704545

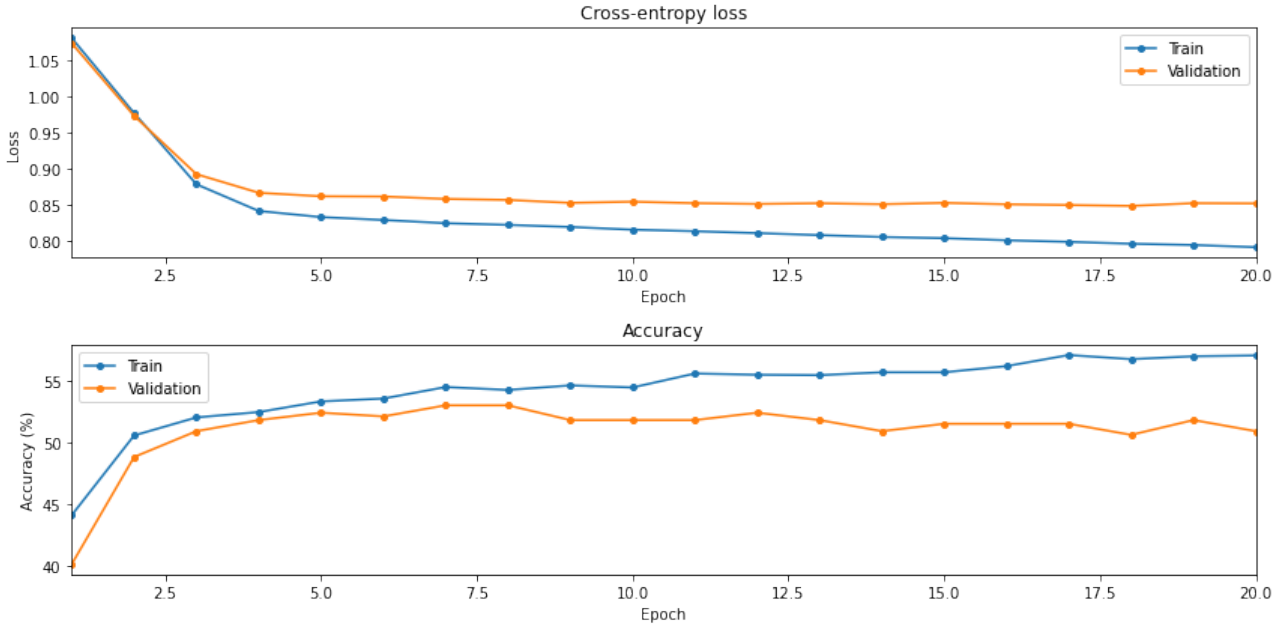


Figure 6.4: TT-RNN model's cross-entropy loss and accuracy for training and validation datasets

The relative modal importance of the financial data can be interpreted by judging which data modality had the largest changes during training [23]. The normalized TT-core change over the training is defined as follows [23]:

$$\frac{||\Delta \mathcal{G}_n^e||_F^2}{I_n J_n R_{n-1} R_n}, \tag{6.1}$$

where $\Delta \mathcal{G}_n^e = \mathcal{G}_n^e - \mathcal{G}_n^{e-1}$ and e refers to epoch ($e = 2, \ldots, 20$). The changes in the core weights is illustrated in figure 6.5. From the figure it can be seen that cores 1 and 5 experienced the greatest changes. This implies that the mode-1 and mode-5 are responsible for most of the predictive power of the TT-RNN model.

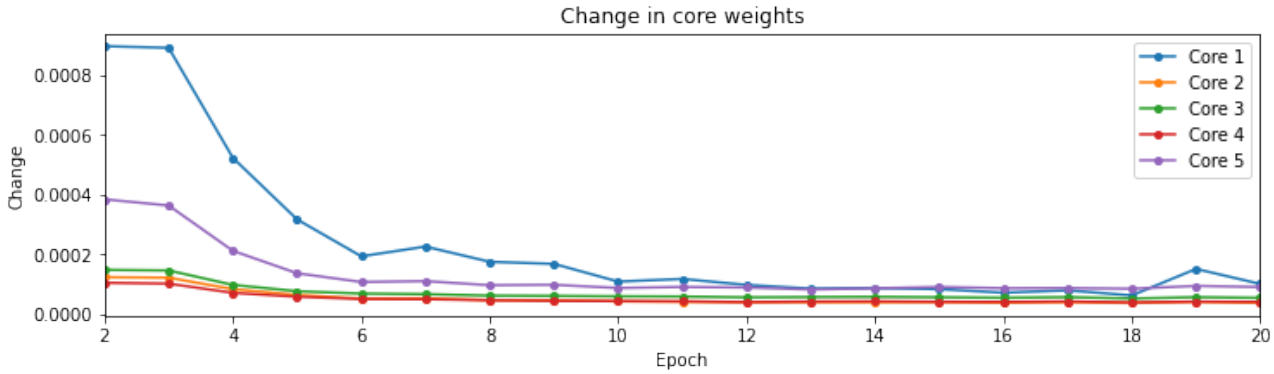


Figure 6.5: TT-RNN model's changes in core weights

6.2.3 Trading

Table 6.5 shows the trading performance of the TT-RNN model. From the table it can be seen that the model achieved total returns of around 25% which is a vast improvement compared with the RNN model. Additionally, all of the other trading performance metrics are better than the RNN model's. The improvement in the trading performance metrics comes even with improved complexity as the model used less parameters than the RNN model. Figure 6.6 illustrates the actual long-only and trading returns by the TT-RNN model of the SPX. From the figure it can clearly be seen that the trading returns of the model beat the actual long-only returns for most of 2020. However, as the global economy started to recover after the COVID-19 pandemic due to strong economy recovery actions, the actual returns started to catch up the model's trading returns and eventually became greater.

Table 6.5: TT-RNN model's trading results

Metric	**Value**
Total return (%)	24.770336
Average return (%)	0.070370
Maximum drawdown (%)	10.842586
Maximum drawdown duration	38
Sharpe ratio	0.072245
Skew	1.198410
Kurtosis	10.044279

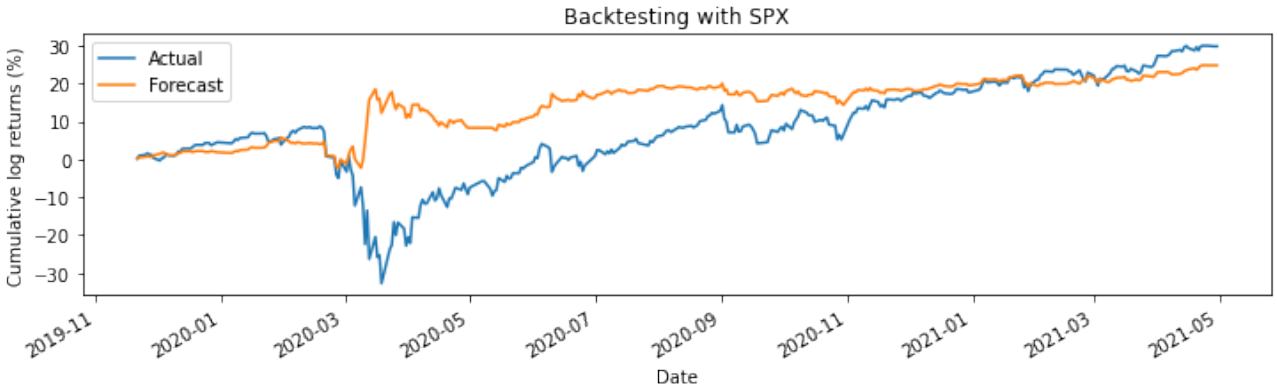


Figure 6.6: TT-RNN model's backtesting with the SPX

6.3 Hankel Tensor Tensor-Train Recurrent Neural Network Model Results

Table 6.6 displays the trading results for each of the H-TT-RNN models of varying TT-ranks. From the table it can be seen that TT-ranks of 1 and 9 achieved the best results. Furthermore, the model having TT-ranks of 1 outperformed the model having TT-ranks of 9 in terms of total return, average return, maximum drawdown, maximum drawdown duration and Sharpe ratio. As such the rest of this section will focus on the H-TT-RNN model having TT-ranks of 1.

Table 6.6: H-TT-RNN model's trading results for varying TT-ranks. The best results are bolded.

Metric	TT-ranks									
	1	2	3	4	5	6	7	8	9	10
Total return (%)	**20.631500**	-21.712922	-10.101960	-18.451957	15.121742	-27.803695	-45.096299	3.871301	16.123210	6.022994
Average return (%)	**0.066769**	-0.070268	-0.032692	-0.059715	0.048938	-0.089980	-0.145943	0.012528	0.052179	0.019492
Maximum drawdown (%)	**11.369877**	28.340459	19.032745	31.945689	12.527364	32.513842	46.248532	23.199347	13.553643	28.685106
Maximum drawdown duration	**5**	253	214	135	103	21	276	16	113	114
Sharpe ratio	**0.059959**	-0.053505	-0.034866	-0.049625	0.044222	-0.082932	-0.130778	0.011790	0.046798	0.016791
Skew	-0.663411	-1.621748	0.395219	-0.851207	0.836208	-1.264709	-1.033108	-0.165497	**1.149007**	0.593754
Kurtosis	9.261731	16.669796	8.248177	13.120155	10.910543	9.150398	9.100860	10.896925	**8.206052**	12.191780

6.3.1 Complexity

Since the H-TT-RNN model used TT-ranks of 1, the number of parameters by the weight matrix $\mathbf{W}_x$, 184320 parameters, was replaced by only 64 parameters. This means a reduction by a factor of 2880. The total number of parameters used by the network was 1052739 parameters.

6.3.2 Training and Testing

Table 6.7 shows the losses and accuracies of the H-TT-RNN model. The achieved test accuracy of 52.427184% saw a further improvement from the test accuracy of the TT-RNN model (51.704545%). Figure 6.7 shows the evolution of the loss and accuracy curves of the H-TT-RNN model for train and validation datasets. Focusing on the accuracy curves, it can be seen that they appear to diverge over the epochs. This could imply that the model is slightly over-fitting despite its increased accuracy.

Figure 6.8 shows changes of the core weights over the epochs. From the figure it can be seen that core 1 experienced the greatest changes whereas rest of the cores experienced smaller changes.

Table 6.7: H-TT-RNN model's loss and accuracy results

Metric	Value
Train loss	0.797222
Validation loss	0.825357
Test Loss	0.840815
Train accuracy (%)	59.944506
Validation accuracy (%)	52.095808
Test accuracy (%)	52.427185

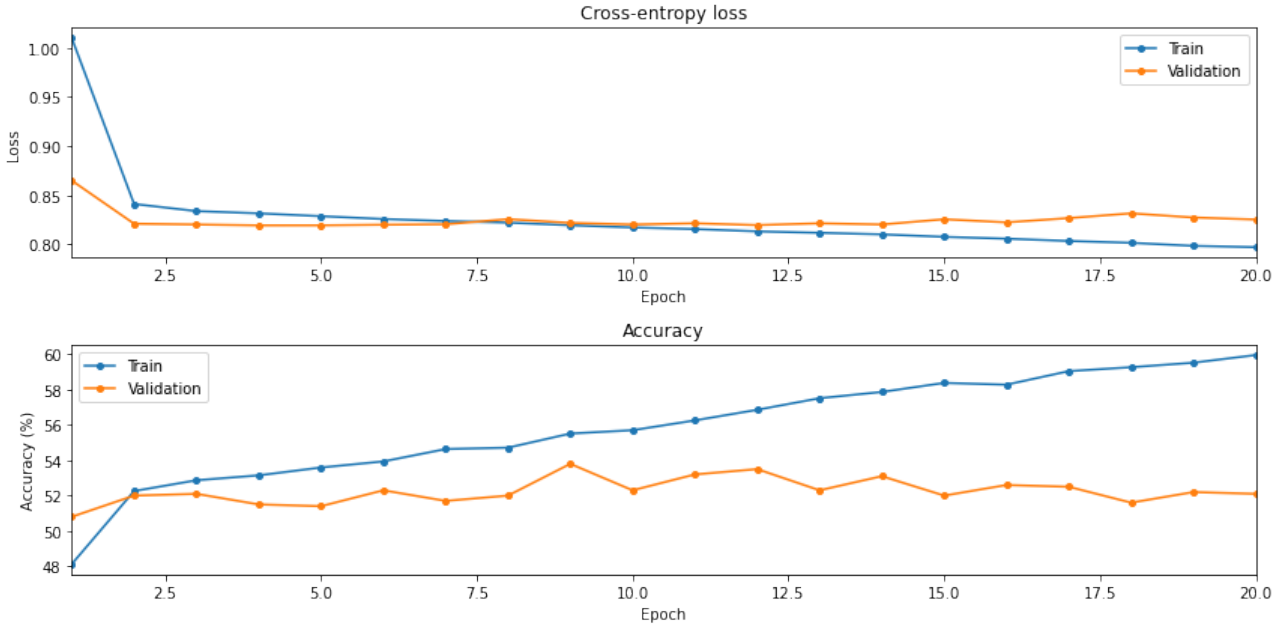


Figure 6.7: H-TT-RNN model's cross-entropy loss and accuracy for training and validation datasets

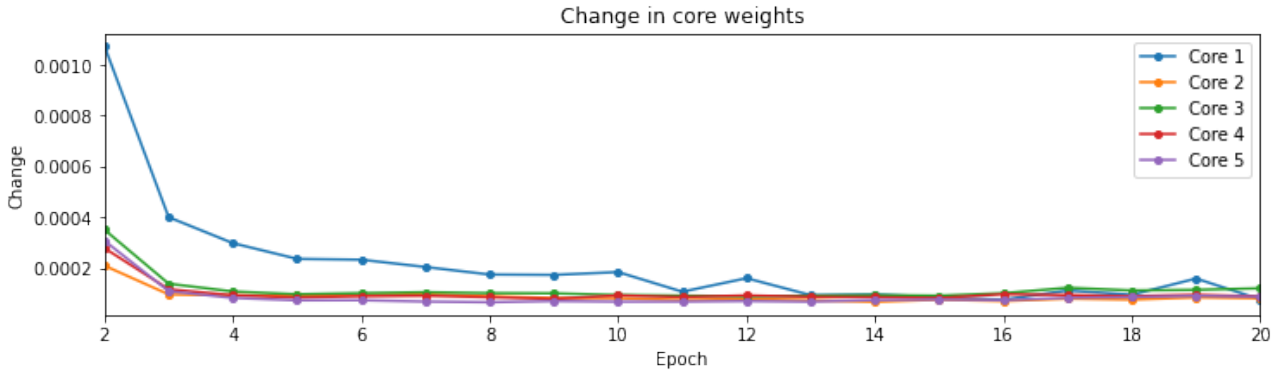


Figure 6.8: H-TT-RNN model's changes in core weights

This means that mode-1 is responsible for most of the predictive power of the H-TT-RNN model.

6.3.3 Trading

Table 6.8 shows the trading performance of the H-TT-RNN model. The model outperformed the standard RNN model in all of the metrics but fell short to the TT-RNN model except for the maximum drawdown duration kurtosis values. Figure 6.9 illustrates the actual long-only and trading returns by the H-TT-RNN model of the SPX. It can be seen from the figure that the H-TT-RNN model outperformed the long-only returns for most of the testing range. Only in April 2021 the long-only cumulative returns became greater than the trading returns.

Table 6.8: H-TT-RNN model's trading results

Metric	Value
Total return (%)	20.631494
Average return (%)	0.066769
Maximum drawdown (%)	11.369877
Maximum drawdown duration	5
Sharpe ratio	0.059959
Skew	-0.663411
Kurtosis	9.261732

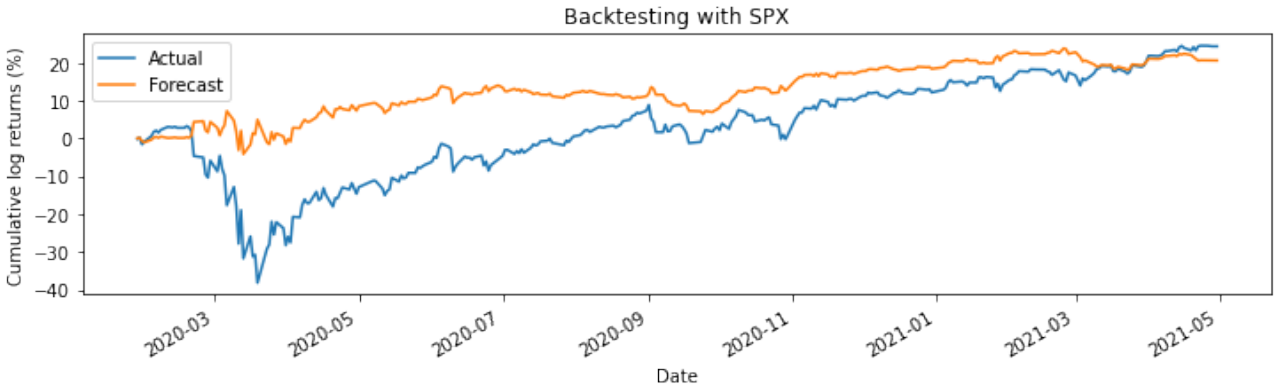


Figure 6.9: H-TT-RNN model's backtesting with the SPX

6.4 Comparison of Results

Table 6.9 shows a summary of the complexities of the models in terms of the number of parameters used by the networks. It can be seen that the H-TT-RNN model required the fewest number of parameters out of the models. The difference in the number of parameters between the TT-RNN and H-TT-RNN models derives from their optimal TT-ranks which were obtained to be 3 and 1, respectively. However, it should be noted that despite the optimal H-TT-RNN model requiring the fewest number of parameters, its unofficial training time took the longest by a large margin.

Table 6.9: Complexities of the models summarized. The best value is bolded.

	Model		
	RNN	TT-RNN	H-TT-RNN
Number of parameters	1236995	1053155	**1052739**

Table 6.10 summarizes the losses and accuracies for the models. Focusing on the test losses and test accuracies, it can be seen from the table that the H-TT-RNN model outperformed the RNN and TT-RNN models and that the TT-RNN model outperformed the RNN model. This result is in line with the results presented in [8] where Hankelizing the input tensor resulted in a better forecasting accuracy. Moreover, it is notable that by applying TT decomposition in the NN, the test results for the compressed models (H-TT-RNN and TT-RNN) were better

Table 6.10: Losses and accuracies of the models summarized. The best results are bolded.

Metric	Model		
	RNN	TT-RNN	H-TT-RNN
Train loss	0.807256	**0.791837**	0.797222
Validation loss	1.163723	0.852382	**0.825357**
Test loss	1.250395	0.887323	**0.840815**
Train accuracy (%)	**61.863559**	57.038269	59.944506
Validation accuracy (%)	34.131736	50.898202	**52.095808**
Test accuracy (%)	23.579545	51.704545	**52.427185**

while having a lower complexity when compared with the uncompressed standard RNN model.

Table 6.11 shows a summary of the various trading metrics for the models. From the table it can be seen that the TT-RNN model had the best results in five out of seven metrics with the H-TT-RNN model having the best results in the rest of the metrics. However, all in all the trading results by the TT-RNN and H-TT-RNN models were similar to each other.

Table 6.11: Trading performances of the models summarized. The best results are bolded.

Metric	Model		
	RNN	TT-RNN	H-TT-RNN
Total return (%)	-12.887654	**24.770336**	20.631494
Average return (%)	-0.036613	**0.070370**	0.066769
Maximum drawdown (%)	28.793144	**10.842586**	11.369877
Maximum drawdown duration	81	38	**5**
Sharpe ratio	-0.038219	**0.072245**	0.059959
Skew	-0.673402	**1.198410**	-0.663411
Kurtosis	16.117341	10.044279	**9.261732**

Chapter 7

Conclusions and Future Work

7.1 Evaluation

This project was concerned with a big data approach to financial forecasting through tensors. In doing so, the focus was on different tensorization methods and how they could be applied in financial forecasting by employing a ML based approach through RNNs. Different tensorization methods were first explored in chapter 2 after which examples of the tensorization methods were presented in chapter 3. Ultimately, the scope for the empirical work was limited to only Hankelization as it showed the most promise. This choice was further motivated by the use of Hankelization in [8] where it was used in multiple short non-financial time series forecasting.

To my knowledge, this project was the first one in which Hankelization was combined with TT decomposition and RNN to forecast in a financial context. The precise research question was how effective such an approach is. To answer the research question, three assets from four different asset classes were chosen with the goal being to forecast the next day returns of the SPX. The H-TT-RNN model was trained and tested with 15 years worth of data from 2006 to 2021 and compared with a standard uncompressed RNN as well as TT-RNN models. The simulation results showed that the H-TT-RNN model had the best test loss and accuracy results out of all of the models. A similar result was obtained in [8]. Furthermore, by applying TT decomposition, the H-TT-RNN and TT-RNN models required fewer number of parameters

when compared with the RNN model. This means that the use of TT decomposition compressed the networks while offering regularization properties. This allows for a better physical interpretability in terms of the relative modal importance in a financial context and helps to mitigate the black-box criticism of NNs [23].

All in all, this project met its goals in exploring different tensorization methods in a financial context while employing a big data approach. Moreover, both the theoretical and empirical work support the finding that by Hankelizing the input tensor while using TT decomposition and RNN, the forecasting accuracy in a financial context can be further improved.

7.2 Future Work

The work done in this project could be extended in several ways and possible directions for future work have been discussed throughout this report. These are summarized next.

First, different tensorization methods could be employed in a financial forecasting setting. In this project the scope was limited to only Hankelization but there are still several more tensorization methods outlined in [13] that could be explored further.

Second, this project used TT decomposition to offer compression and regularization properties for the network. However, there are other TD methods available that could be used as well. For example, in [8] Tucker decomposition was used in ARIMA forecasting. Instead of ARIMA forecasting, Hankelization combined with, for example, Tucker decomposition and RNN in financial forecasting gives an interesting direction for future work.

Finally, more sophisticated RNN models such as LSTM and GRU models could be used. These tackle the inherent limitation of RNNs regarding their vanishing/exploding gradients. Moreover, instead of using a ML based forecasting method, ARIMA and GARCH forecasting methods (which were explained in detail in chapter 2) could be used in financial forecasting combined with different tensorization and TD methods. This would also eliminate the black-box nature inherent to NNs since the ARIMA and GARCH methods are readily more interpretable.

Bibliography

[1] K. Zhang, X. Zhang, and Z. Zhang, "Tucker Tensor Decomposition on FPGA," 2019, 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 1–8.

[2] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. Mandic, "Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 1 Low-Rank Tensor Decompositions," 2016, Foundations and Trends in Machine Learning, 9(4-5):249–429.

[3] Y. Yang, D. Krompass, and V. Tresp, "Tensor-Train Recurrent Neural Networks for Video Classification," 2017, ICML.

[4] M. Rei and J. Wang, "Introduction to Machine Learning," 2020, Class Lecture, Department of Computing, Imperial College London.

[5] B. Ramsundar and R. B. Zadeh, "TensorFlow for Deep Learning," 2018, O'Reilly Media, https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html, accessed on 21.5.2021.

[6] Y. Li, "Deep Learning," 2021, Class Lecture, Department of Computing, Imperial College London.

[7] T. Yokota, B. Erem, S. Guler, S. K. Warfield, and H. Hontani, "Missing Slice Recovery for Tensors Using a Low-Rank Model in Embedded Space," 2018, IEEE/CVF Conference on Computer Vision and Pattern Recognition, 8251–8259.

[8] Q. Shi, J. Yin, J. Cai, A. Cichocki, T. Yokota, L. Chen, M. Yuan, and J. Zeng, "Block Hankel Tensor ARIMA for Multiple Short Time Series Forecasting," 2020, Proceedings of AAAI.

[9] N. S. Kuusisto, "Forecasting Methods Applied to Macroeconomic Variables," 2021, Bachelor's Thesis, Department of Economics, Aalto University School of Business.

[10] L. Kuang, F. Hao, L. T. Yang, M. Lin, C. Luo, and G. Min, "A Tensor-Based Approach for Big Data Representation and Dimensionality Reduction," 2014, IEEE Transactions on Emerging Topics in Computing, 2:280–291.

[11] D. P. Mandic, I. Kisil, and G. Calvi, "Tensor Analysis and Big Data," 2019, Class Lecture, Signal Processing and Machine Learning for Finance, Department of Electrical and Electronic Engineering, Imperial College London.

[12] A. Spelta, "Financial market predictability with tensor decomposition and links forecast," 2017, Applied Network Science, 2(7).

[13] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, I. V. Oseledets, M. Sugiyama, and D. Mandic, "Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 2 Applications and Future Perspectives," 2017, Foundations and Trends in Machine Learning, 9(6):431–673.

[14] K. jae Kim, "Financial time series forecasting using support vector machines," 2003, Neurocomputing, 55:307–319.

[15] M. K. Okasha, "Using Support Vector Machines in Financial Time Series Forecasting," 2014, International Journal of Statistics and Applications, 4(1):28–39.

[16] G. G. Calvi, V. Lucic, and D. P. Mandic, "Support Tensor Machine for Financial Forecasting," 2019, ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 8152–8156.

[17] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," 2020, International Journal of Forecasting, 36(3):1181–1191.

[18] M. Nabipour, P. Nayyeri, H. Jabani, A. Mosavi, E. Salwana, and S. Shahab, "Deep Learning for Stock Market Prediction," 2020, Entropy, 22(8):840.

[19] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, "Tensorizing Neural Networks," 2015, Advances in Neural Information Processing Systems, 28.

[20] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets," 2017, 2013 IEEE International Conference on Acoustics, Speech and Signal Processing.

[21] X. Cao and G. Rabusseau, "Tensor Regression Networks with various Low-Rank Tensor Approximations," 2017, .

[22] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition," 2014, .

[23] Y. L. Xu, G. G. Calvi, and D. P. Mandic, "Tensor-Train Recurrent Neural Networks for Interpretable Multi-Way Financial Forecasting," 2021, arXiv e-prints.

[24] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," 2015, IEEE Signal Processing Magazine, 32(2):145–163.

[25] M. Mørup, "Applications of tensor (multiway array) factorizations and decompositions in data mining," 2011, Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery, 1(1):24–40.

[26] O. Debals and L. De Lathauwer, "The Concept of Tensorization," 2017.

[27] Q. Zhang, M. Berry, B. Lamb, and T. Samuel, "A parallel nonnegative tensor factorization algorithm for mining global climate data," 2009, Proceedings of the International Conference on Computational Science (ICCS), 405–415.

[28] J.-M. Papy, L. De Lathauwer, and S. Van Huffel, "Exponential data fitting using multilinear algebra: The single-channel and multi-channel case," 2005, Numerical Linear Algebra with Applications, 12(8):809–826.

[29] O. Debals, M. Van Barel, and L. De Lathauwer, "Löwner-Based Blind Signal Separation of Rational Functions With Applications," 2015, IEEE Transactions on Signal Processing, 64(8):1909–1918.

[30] S. Chen and S. A. Billings, "Representations of non-linear systems: the NARMAX model," 1989, International Journal of Control, 49(3):1013–1032.

[31] D. A. Vaccari, "Taylorfit MPR," 2003, Simetrica, LLC, http://www.simetrica-llc.com/Products/MPR/index.html.

[32] L. Hardesty, "Explained: Neural networks," 2017, MIT News, https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414, accessed on 21.5.2021.

[33] B. Kainz, "Deep Learning," 2021, Class Lecture, Department of Computing, Imperial College London.

[34] N. Winovich, "Deep Learning Optimization," 2021, Purdue University, https://www.math.purdue.edu/ nwinovic/deep_learning_optimization.html, accessed on 22.5.2021.

[35] C. Liu, S. C. H. Hoi, P. Zhao, and J. Sun, "Online ARIMA algorithms for time series prediction," 2016, Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 1867–1873.

[36] E. Stellwagen and L. Tashman, "ARIMA: The Models of Box and Jenkins," pp. 28–33, 2013, Foresight: The International Journal of Applied Forecasting, 30:28–33.

[37] F. X. Diebold, "Econometric Data Science: A Predictive Modeling Approach," 2019, Department of Economics, University of Pennsylvania, https://www.sas.upenn.edu/ fdiebold/Textbooks.html, accessed on 3.5.2021.

[38] ——, "Forecasting in Economics, Business, Finance and Beyond," 2017, Department of Economics, University of Pennsylvania, https://www.sas.upenn.edu/ fdiebold/Textbooks.html, accessed on 1.4.2021.

[39] E. Sukhanova, S. Shirnaeva, and A. Mokronosov, "Econometric Models for Forecasting of Macroeconomic Indices," 2016, international Journal of Environmental & Science Education, 11(16):9191–9205.

[40] D. Ruppert and D. S. Matteson, "Statistics and Data Analysis for Financial Engineering," 2015, Statistics and Data Analysis for Financial Engineering. Springer Texts in Statistics. Springer, New York, NY. https://doi.org/10.1007/978-1-4939-2614-5_14.

[41] W. W. S. Wei, "Multivariate Time Series Analysis and Applications," 2019, John Wiley Sons Ltd.

[42] A. Ganti, "Asset Class," 2021, Investopedia, https://www.investopedia.com/terms/a/assetclasses.asp, accessed on 12.5.2021.

[43] J. Chen, "Cash Equivalents," 2020, Investopedia, https://www.investopedia.com/terms/c/cashequivalents.asp, accessed on 12.5.2021.

[44] Plus500, "What are the most traded commodities?" 2021, https://www.plus500.com/Trading/Commodities/What-Are-the-Most-Traded-Commodities 2, accessed on 26.5.2021.

[45] H. C. Bjørnland, "VAR Models in Macroeconomic Research," 2000, statistics Norway, Research Department, 48.

[46] Investing.com, 2021, Database, https://www.investing.com, accessed on 25.5.2021.

[47] N. S. Kuusisto, "Signal Processing and Machine Learning for Finance," 2021, Coursework, Signal Processing and Machine Learning for Finance, Department of Electrical and Electronic Engineering, Imperial College London.

[48] M. López de Prado, "The 10 Reasons Most Machine Learning Funds Fail," 2018, Journal of Portfolio Management, Forthcoming.

[49] D. P. Mandic, "Introduction: Background, Definitions and Objectives," 2021, Class Lecture, Signal Processing and Machine Learning for Finance, Department of Electrical and Electronic Engineering, Imperial College London.

[50] Y. L. Xu, "Tensor-Train Recurrent Neural Networks in Financial Forecasting," 2019, Final Year Project, Department of Electrical and Electronic Engineering, Imperial College London.

[51] M. Gupta, "Feature Scaling – Part 2," 2019, GeeksforGeeks, https://www.geeksforgeeks.org/ml-feature-scaling-part-2/, accessed on 14.6.2021.

[52] R. Vashisht, "When to perform a Feature Scaling?" 2021, Atoti, https://www.atoti.io/when-to-perform-a-feature-scaling/, accessed on 1.6.2021.

[53] J. Fernando, "Sharpe Ratio," 2021, Investopedia, https://www.investopedia.com/terms/s/sharperatio.asp, accessed on 30.5.2021.

Appendix

A Ethical, Legal and Safety Considerations

Ethical, legal and safety considerations of the project work are considered in this section.

A.1 Ethical Plan

Imperial College Research Ethics defines ethical acting as follows. Being 'ethical' means acting in accordance with a set of core values and principles, in particular, integrity, compliance with the law, respect for human rights and avoiding unnecessary risk to people's safety and well-being. Imperial College London seeks to ensure that any potential ethical risks arising from research are limited strictly in proportion to the importance of the intended benefits.

A researcher must, therefore, consider the ethical implications of any work that:

- has the potential to damage the mental or physical health of human participants, (e.g. volunteers, College staff and students, or patients,) or others who may be affected
- has the potential to jeopardise the safety and liberty of people affected by the research (e.g. volunteers working in sensitive situations or abroad)
- has the potential to compromise the privacy of individuals whose data is involved in the work

- involves methods (e.g. genetic research, interviews, questionnaires, randomised control trial) or subject matter (e.g. recreational and controlled drugs, human impact on the environment) that are sensitive and therefore need to be managed consistently with the College's high public reputation
- carries a risk of an actual or perceived conflict of interest on the part of researchers and/or the College

Considering the above, the project work is not expected to raise ethical issues as long as the guidelines are followed with care.

A.2 Legal Plan

There are not any expected legal considerations regarding the project work. However, care needs to be taken with the data used as it is provided by Yahoo Finance/Bloomberg Terminal. Thus, their terms and conditions regarding the use of data need to be obeyed.

A.3 Safety Plan

The Safety Assessment on the project web page has been carried out. Care needs to be taken with Data Infrastructure Safety. This applicable to the project work because data needs to be retrieved. Reliable sources such as Yahoo Finance are going to be used to retrieve data as they are unlikely to compromise any IT systems through, for example, the spreading of viruses or improper use of networks.

None of the other safety issues (electrical safety, physical safety, chemical safety, fire safety, biological safety, animal safety, appliance safety, airspace safety or study participant safety) are of concern in the project work. This is because the project consists solely of analytical work followed by implementations of the tensorization and forecasting methods, thus eliminating the other possible safety issues.

B Code for Empirical Part

The code is available at *https://github.com/niklaskuusisto/Final-Year-Project*. Note that the implementation used a version 2.2.4 of Keras and 1.15.5 of TensorFlow. It has not been tested with other versions.

The code written for this project referenced the works by [50], [23], [8], [3], [19].