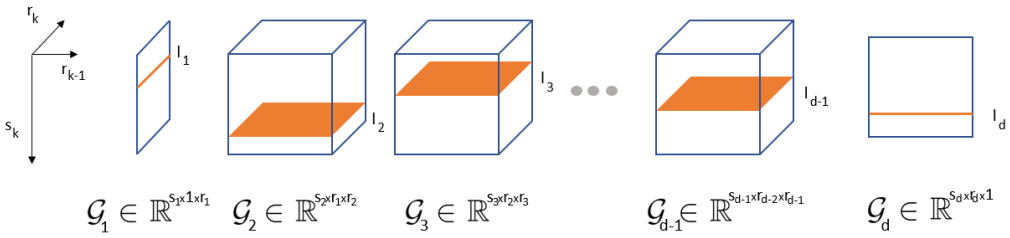
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2019



Project Title:	**Tensor-Train Recurrent Neural Networks in Financial Forecasting**
Student:	**Yao Lei Xu**
CID:	**01062231**
Course:	**4T**
Project Supervisor:	**Professor Danilo Mandic**
Second Marker:	**Professor Athanassios Manikas**

Abstract

This project investigates the application of tensor-train decomposition (TTD) in recurrent neural networks for forecasting high-dimensional financial time series. The mathematical background is provided for various tensor operations and neural network architectures, discussing their advantages and disadvantages in terms of complexity and modelling power. Subsequently, the project derives the tensor-train neural network architectures mathematically and discusses the practical algorithmic implementation. Furthermore, the interpretability of TTD based neural networks is discussed by exploring the links that the TTD establishes between the data structure and the neural network models. The derived models are then applied to financial data to forecast the returns of chosen assets, which are evaluated in terms of model performance, model complexity and trading performance. Overall, TTD based models can super-compress standard neural network architectures, reaching a compression factor of $21\text{x}10^3$ in storage complexity with little loss in performance. The model also demonstrates significant regularization power and better generalization properties due to less over-fitting. Finally, the experimental results and possible future research directions are discussed.

Acknowledgements

I would like to express my sincere gratitude to Professor Danilo P. Mandic for providing me with guidance and feedback.

I would also like to thank Giuseppe G. Calvi for offering invaluable help, discussion and advice for this project.

Finally, I would like to thank my family for their unconditional love and support, as well as my friends for being my source of inspiration over the past 4 years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Machine learning is the study of algorithms and statistical models which aims to identify underlying patterns present in the data. Artificial neural networks are a particular family of machine learning models inspired by the structure of biological neurons. Thanks to modern computing capabilities and increasing amount of data, modern neural networks are wider and deeper than ever before and have achieved state-of-the-art performance in many tasks, with recurrent neural networks taking the crown in terms of time series analysis.

However, as the world enters the era of Big Data, multi-dimensional data are being produced on a unprecedented level, which poses significant challenges to classical data analysis techniques. Many problems now require data solutions that can work not only in large volume, but are also fast and robust to noisy and incomplete data. These challenges are particularly problematic for neural network models, since they are known for being easily over-fit and need high training memory and time requirements [2].

Fortunately, the field of tensor decomposition offers a solution to the high-volume, high-velocity, high-variety and high-veracity challenges of Big Data (figure 1.1) [1]. Tensors are multi-linear generalization of vectors and matrices which preserve data structures, while tensor decomposi-

tion methods aims to represent multi-modal tensors in specific formats. Tensor decomposition methods can compresses the data, allows faster algebraic operations and establishes links within the data structure which makes the data interpretable.

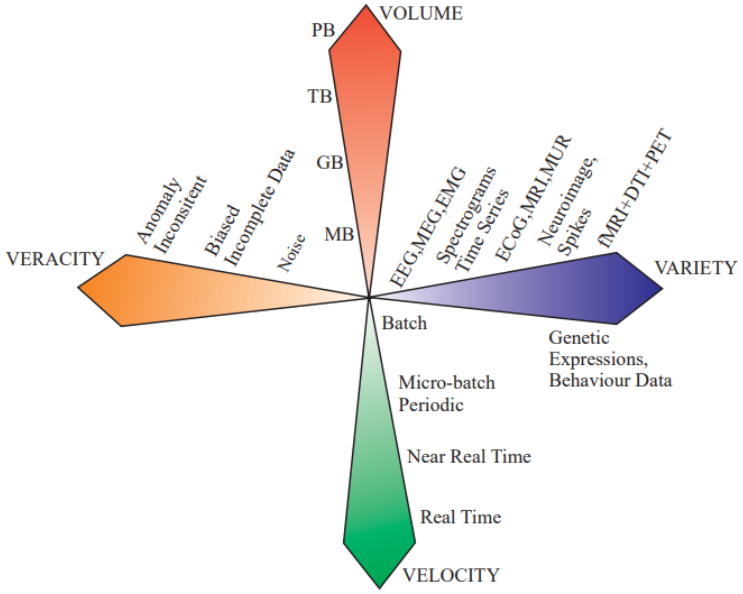


Figure 1.1: Challenges of Big Data [1]

Specifically, the tensor-train (TT) decomposition can efficiently represent high-dimensional tensors in very few parameters. Compared to other decomposition methods, the TT format offers super-compression capabilities [3], does not inherently suffer from the curse of dimensionality and has many numerically stable algorithms for tensor operations [4]. Due to its computational advantages, the TT format has found several applications in fields of big data and machine learning, offering promising results. In particular, the tolerance to the curse of dimensionality allows the training of TT based recurrent neural networks (RNN) in an end-to-end fashion for high-dimensional data [5], which has always been a major issue in the field. In addition, tensor decomposition in neural networks has shown powerful regularization properties and improves their generalization capabilities [6].

In terms of application, the project will focus on financial data which are natural high-dimensional tensors (figure 1.2). Financial markets generate many features over time (bid prices, ask prices, volume, fundamentals, sentiment, etc.) across a large number of asset classes. However, since these data exist only for a limited amount of time, the coupling of data scarcity and high-

dimensionality makes financial data susceptible to the curse of dimensionality. The large number of variables makes the dataset unsuitable for deep-learning methods, especially for RNN models since the complexity and the number of parameters needed for modelling grows drastically with the number of dimensions, making it prone to over-fitting. Finally, TT format can increase interpretability of the data, which is of paramount importance in finance.

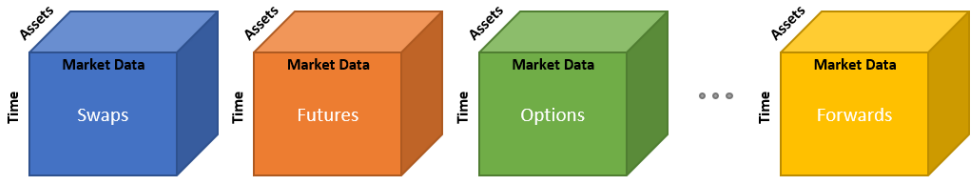


Figure 1.2: Financial Data Tensor

In summary, the project aims to create a tensor-train based recurrent neural network (TT-RNN) for high-dimensional financial forecasting. The performance will be analyzed and compared to classical neural networks in terms of model complexity, model performance and trading performance. In addition, the project will explore different ways of interpreting the model via the TT format to overcome the black-box nature of neural networks. To this end, the main body of the report is structured as follows:

- *Background Theory* - provides the mathematical background relating to tensors and neural networks
- *Tensor-Train Neural Networks* - presents the mathematical framework for TT neural networks and discusses its algorithmic implementation.
- *Financial Application* - applies the final model to forecast financial data.
- *Conclusion and Future Work* - discusses the results of various experiments and proposes future research directions.

1.2 Related Work

1.2.1 Financial Forecasting

Financial forecasting has been an important topic of research in both academia and industry since the correct prediction of asset movements offers significant economic benefits. In addition to classical models governed by analytic methods such as the Black Scholes equation [7], a large amount of research has been done using more data-driven models such as support vector machines (SVM) [8] and neural networks (NN) [9].

SVM models are different from most machine learning methods since they minimize the structural risk instead of the empirical risk and work better in high dimensional settings. SVMs can effectively control the over-fitting of the model and are robust to noise. However, in terms of modelling power, SVMs often lag in performance compared to NN based models.

NN based approaches have been used for financial forecasting since the 20th century [9] and are praised for their powerful modelling capabilities. Recently, modern methods from natural language processing using recurrent neural networks have found success in financial forecasting [10], which are more powerful than the traditional fully-connected NNs. However, the major drawback of NN based models is that they are black-boxes that are hard to interpret and can easily over-fit.

Overall, TT decomposition can potentially solve all these issues by acting as a regularizer for the highly expressive NN models and establishing links across the structure of high-dimensional data that increases interpretability.

1.2.2 Neural Networks and High-Dimensional Modelling

Neural networks have achieved state of the art results in many fields thanks to the recent advances in computational power and GPU computing. However, the main drawback of these models is that they require a large amount of data to be trained properly, which is highly

consuming in terms of time and memory. In addition, when the input data is high-dimensional in nature (such as financial and RGB video data), the number of hidden units grows drastically, which makes it susceptible to the curse of dimensionality and makes regularization methods more computationally expensive. There are several methods for reducing the complexity of high-dimensional data modelling, such as using dimensionality reduction methods (e.g. PCA) [11] [12], features extraction methods using other machine learning models [13] [14] and layer compression via low-rank computation of the weight matrix [15].

The dimensionality reduction of the input features does speed up the training process, but the features projection and reduction often eliminates essential information for further modelling (such as discriminative information for classification) since they are unsupervised methods. Features extraction with other machine learning models can create a much more compact low-dimensional representation of information, but this is often trained separately (hence sub-optimally) for further high-dimensional data modelling. Low-rank compression methods however have demonstrated great reduction in storage complexity (30-50% less in [15]) with little loss in performance.

Since the TTD is a compression method with great complexity characteristics, the results from other papers motivate the use of the tensor decomposition as a way of improving deep learning models [16], in particular by using the TT format [17] [5].

1.2.3 Tensor Decomposition Based Neural Networks

Several papers have applied tensor decomposition to neural networks. Firstly, paper [17] introduces the idea of using TTD as a compression tool for fully-connected neural networks. Then, paper [5] generalizes this idea to recurrent neural networks for video classification, demonstrating the possibility of training with high-dimensional data in an end-to-end fashion. Other than its compression capabilities, paper [6] demonstrates the use of various tensor decomposition methods in neural networks for regularization purposes. Finally, similar work have been carried out using other decomposition methods such as CPD [16].

1.2.4 Interpretation of TT Neural Networks

Tensor decomposition has found vast success in deep learning in terms of compression and regularization capabilities. However, relatively few work has been done in terms of interpreting these networks. Most of the existing literature on the theoretical understanding of tensors and deep learning comes from physics, where the tensor-train format has been used to model entanglement properties as matrix product states [18]. However, most of the work has been done in terms of representing neural networks as tensor networks [19] [20] instead of investigating the compression of neural network layers via tensor decomposition.

Chapter 2

Background Theory

2.1 Tensors

2.1.1 Notations and Definitions

Due to the high-dimensionality of mathematical objects treated in this project, it is essential to start with a concise overview of notations. However, this paper will outline only the definitions needed for the application of TTD in neural networks, which is by no means exhaustive. For a full review of tensor notations and operations, please refer to [21].

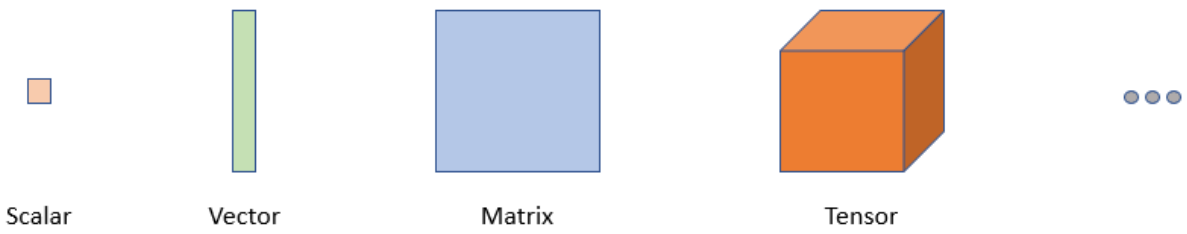


Figure 2.1: Scalar, Vector, Matrix, Tensor

Tensors are multi-linear generalization of vectors and matrices. Mathematically, an order d tensor $\mathcal{A} \in \mathbb{R}^{p_1 x p_2 x \ldots x p_d}$ has d dimensions, where each of the dimension is conventionally referred to as mode. For each of the k^{th} mode where $k = 1, 2, ..., d$, it has a total of p_k elements that can

be indexed using $l_k = 1, 2, ..., p_k$. Therefore, this data structure can be simply regarded as a multi-dimensional array where each element of the tensor is defined with d indices $\mathcal{A}[l_1, ..., l_d]$.

It follows that special cases of tensors include matrices (order-2 tensors), vectors (order-1 tensors) and scalars (order-0 tensors). However, to differentiate among the aforementioned mathematical objects without ambiguity, the notation in the following table will be used for the remaining of the report.

Number of Modes	Object Name	Object Notation	Scalar Realization
1	Vector	$\mathbf{a}$	$\mathbf{a}[l]$
2	Matrix	$\mathbf{A}$	$\mathbf{A}[l_1, l_2]$
≥ 3	Tensor	$\mathcal{A}$	$\mathcal{A}[l_1, ..., l_d]$

Table 2.1: Tensor Notation

Sub-tensors are tensors formed by taking a sub-set of the original tensor. A special class of sub-tensors known as fibers are vectors formed by fixing all indices except one, where the non-fixed index is denoted using " : " (e.g. $\mathcal{A}[:, l_2, ..., l_d]$). Similarly, slices are matrices formed by fixing all indices except two (e.g. $\mathcal{A}[:, :, l_3, ..., l_d]$).

2.1.2 Tensor Operations

A common operation that is unique to tensors is the reshaping operation, which consists of reformatting a tensor to a new tensor of a different order and shape. A special case of reshaping is called mode-k unfolding (or mode-k matricization) which consists of creating a matrix by using mode-k fibers of a tensor as columns. An example of this is illustrated in figure 2.2 where the mode-1 unfolding is performed on an order 3 tensor $\mathcal{A} \in \mathbb{R}^{I \text{x} J \text{x} K}$ which is unfolded along mode-1 to form a matrix $\mathbf{A} \in \mathbb{R}^{I \text{x} JK}$. Another special case of reshaping involves transposing the mode-d matricization of an order d tensor $\mathcal{A} \in \mathbb{R}^{p_1 \text{x} p_2 \text{x} ... \text{x} p_d}$ and is denoted as $\mathcal{A}.\text{reshape}(-1, p_d)$.

Several products can be defined in terms of tensor algebra. For this project, the most important product is the outer product denoted as $\mathcal{C} = \mathcal{A} \circ \mathcal{B}$ and is defined such that the result of the

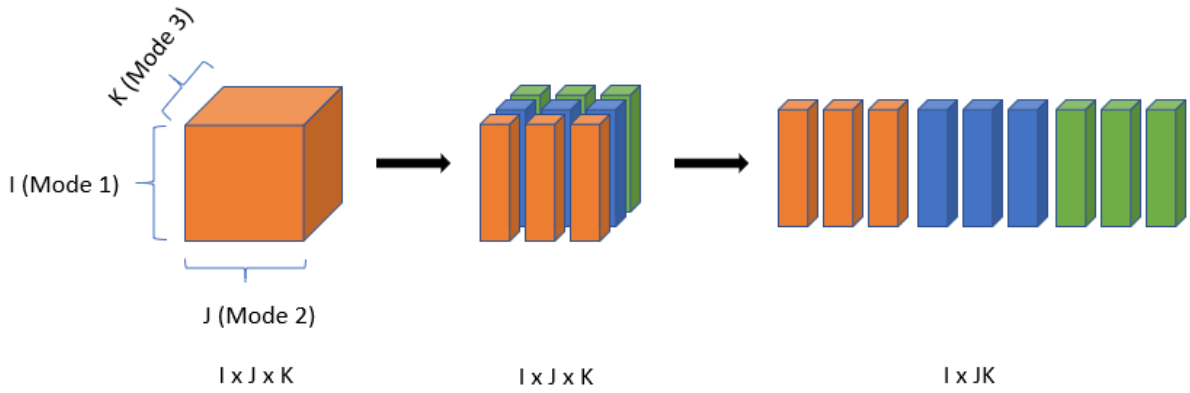


Figure 2.2: Mode-1 Unfolding

multiplication satisfies $\mathcal{C}[i_1, i_2, ..., i_N, j_1, j_2, ..., j_M] = \mathcal{A}[i_1, i_2, ..., i_N]\mathcal{B}[j_1, j_2, ..., j_M]$, where the mathematical objects are $\mathcal{A} \in \mathbb{R}^{I_1 \text{x} I_2 \text{x} ... \text{x} I_N}$, $\mathcal{B} \in \mathbb{R}^{J_1 \text{x} J_2 \text{x} ... \text{x} J_M}$ and $\mathcal{C} \in \mathbb{R}^{I_1 \text{x} I_2 \text{x} ... \text{x} I_N \text{x} J_1 \text{x} J_2 \text{x} ... \text{x} J_M}$. Finally, if the outer product is defined on d vectors $\mathcal{X} = \mathbf{a}_1 \text{o} \mathbf{a}_2 \text{o} ... \text{o} \mathbf{a}_d$ such that $\mathcal{X} \in \mathbb{R}^{p_1 \text{x} p_2 \text{x} ... \text{x} p_d}$ has entries $\mathcal{X}[l_1, l_2, ..., l_d] = a_1[l_1]a_2[l_2]...a_d[l_d]$, then the tensor $\mathcal{X}$ is said to be rank-1.

2.1.3 Decomposition Methods

Several tensor decomposition methods exist, such as the canonical polyadic decomposition (CPD) which represents a tensor as a linear combination of rank-1 tensors [21], which is shown in the equation below.

$$\mathcal{X} = \sum_{r=1}^{R} \lambda_r \mathbf{b}_r^{(1)} \text{o} \mathbf{b}_r^{(2)} \text{o} ... \text{o} \mathbf{b}_r^{(N)} \tag{2.1}$$

The Tucker decomposition (TKD) instead decomposes the original tensor as a multi-linear transformation of a core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \text{x} R_2 \text{x} ... \text{x} R_N}$ using factor matrices $\mathbf{B}^{(n)} = |\mathbf{b}_1^{(n)}, \mathbf{b}_2^{(n)}, ..., \mathbf{b}_{R_n}^{(n)}| \in \mathbb{R}^{I_n \text{x} R_n}$ where $n = 1, 2, ..., N$. This decomposition is represented by equation 2.2 [21].

$$\mathcal{X} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} ... \sum_{r_N=1}^{R_N} \mathcal{G}[r_1, r_2, ..., r_n](\mathbf{b}_{r_1}^{(1)} \text{o} \mathbf{b}_{r_2}^{(2)} \text{o} ... \text{o} \mathbf{b}_{r_N}^{(N)}) \tag{2.2}$$

Finally, the tensor-train decomposition (TTD) efficiently represents a tensor in the tensor train format, where each scalar element of an order d tensor $\mathcal{A} \in \mathbb{R}^{p_1 \text{x} p_2 \text{x} ... \text{x} p_d}$ can be represented by a

non-unique series of matrix multiplications (multiple TT-representation exist for a given tensor [4]) as shown in equation 2.3, where $\mathcal{G}_k$ of k^{th} mode are order 3 tensors called TT-cores and are of size (s_k, r_{k-1}, r_k).

$$\mathcal{A}[l_1, ..., l_d] = \mathcal{G}_1[l_1, :, :]...\mathcal{G}_d[l_d, :, :] \tag{2.3}$$

From a numerical computation point of view, it is easier to think of these order 3 cores as a collection of matrices (core slices) needed for matrix multiplication, where each matrix can be indexed by l_k as $\mathcal{G}_k[l_k, :, :]$ and are of shape (r_{k-1}, r_k). The sequence of r_k for $k = 0, ..., d$ are called the TT-ranks and, since the product of the matrices $\mathcal{G}_k[l_k, :, :]$ results in a scalar, r_0 and r_d are restricted to equal to 1 (note that unlike tensors, the ranks index starts from 0 instead of 1). The tensor-train format can be visualized in figure 2.3 [5].

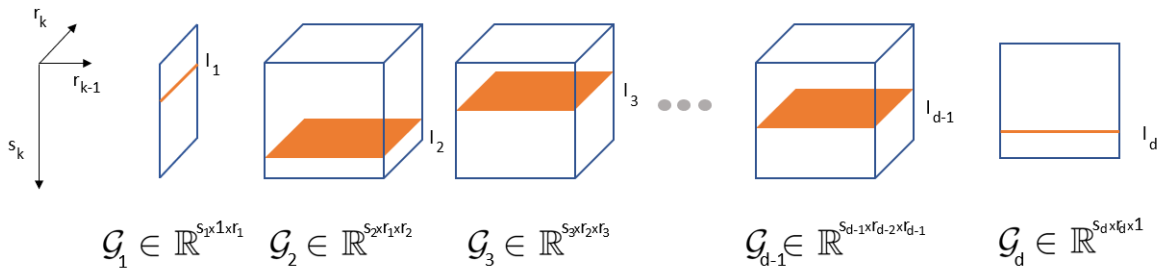


Figure 2.3: TT Matrix Multiplication

2.1.4 Comparison of Decomposition Methods

The above decomposition methods are among the most well-understood ones and have found several applications in engineering. CPD for instance is typically used to find rank-1 factors which are easy to interpret. TKD instead can be constrained in terms of orthogonality and used as a multi-modal extension of the classical PCA [21]. If unconstrained, TKD can also be used simply as a compression tool.

In terms of interpretation, TKD and CPD allows a multi-way analysis of data which is much nicer in terms of uniqueness and assumptions than the classical two-way matrix methods such as ICA (statistical independence constraint). However, in terms of numerical stability and compression, they are far worse than TT methods.

For instance, TT format allows more efficient basic linear algebra operations on tensors such as dot product and matrix-by-vector multiplication and has more stable and fast rounding procedures for decomposition [4] [22]. Most importantly, if a given tensor can be reshaped/quantized into a much higher order tensor $\mathcal{X} \in \mathbb{R}^{q\text{x}q\text{x}\ldots\text{x}q}$ such that q is small, the TT decomposition can achieve the so called *super-compression*, which is a logarithmic reduction of storage requirements [3]: $O(I^d) -> O(d \log_q(I))$. Table 2 compares the compression complexity for an initial storage complexity of $O(I^d)$ for a tensor $\mathcal{X} \in \mathbb{R}^{I_1\text{x}I_2\text{x}\ldots\text{x}I_d}$ where $I = max(I_1, I_2, ..., I_d)$ and R is the highest rank of the tensor decomposition considered. The concept of super-compression is the fundamental idea behind TT neural networks and will be explored in more details later on.

Therefore, the speed, robustness, low memory requirement and tolerance to the curse of dimensionality motivates the use of the TT format in deep-learning domains.

Method	Storage Complexity
CPD	$O(dIR)$
TKD	$O(dIR + R^d)$
TTD	$O(dIR^2)$
Quantized TTD	$O(dR^2 \log_q(I))$

Table 2.2: Storage Complexity Comparison

2.1.5 Tensorization

Now that several tensor operations are outlined, it is important to provide a taxonomy of the tensors that will be treated for the remaining of the paper and how the experiments will relate to each of these. Overall, tensorization of data can be divided in 4 main categories [21]:

1. *Rearrangement of lower-dimensional data structures*: This involves reshaping/quantization of low-dimensional data and is usually done for compression. This is the fundamental idea behind tensorizing neural networks.

2. *Mathematical construction*: This involves mathematical fabrication of additional features. For example, spectral analysis often adds a frequency mode to the original time-series.

3. *Natural tensor data*: Some data are naturally formatted as tensors, such as financial and RGB video data.

4. *Experimental design*: This involves the experimental set ups that have multiple variables such as trials, subjects, conditions etc.

2.2 Neural Networks

2.2.1 Fully-Connected Neural Networks

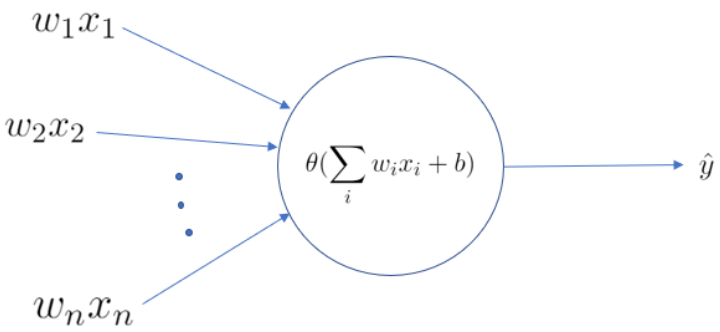


Figure 2.4: Neuron

The basic building block of a neural network is a perceptron which is also known as a neuron (figure 2.4). A basic neuron applies a linear transformation to an M-dimensional input feature vector $\mathbf{x}$ by applying a dot product with the weights $\mathbf{w}$ and adding a scalar bias b to it. Then, a non-linear activation function $\theta(s)$ is applied to its linear mapping, giving it the capability to model non-linear relationships. This function is described in equation 2.4, where $\mathbf{w} \in \mathbb{R}^M$ is a vector and b, $\hat{y}$ are scalars.

$$\hat{y} = \theta(\sum_i w_i x_i + b) = \theta(\mathbf{w}^T\mathbf{x} + b) \tag{2.4}$$

A neural network is formed by connecting multiple neurons together across several layers (figure 2.5). By having a network of multiple neurons, it is possible to model more complex relationships than a just single neuron, provided that the activation functions $\theta(s)$ are not linear for all neurons (since the combination of linear operations results in a linear operation). In addition, the output of the neural network can have as many units as needed depending on the task at hand (e.g. 5 neurons are needed for classification problems with 5 classes using one-hot encoding).

The network of a single fully-connected layer can be described with equation 2.5, which is similar to equation 2.4 but with some change in dimensionality. In this equation, $\mathbf{W} \in \mathbb{R}^{NxM}$ is a matrix (the weight matrix), $\mathbf{b} \in \mathbb{R}^N$ and $\hat{\mathbf{y}} \in \mathbb{R}^N$ are vectors (bias and output vectors respectively), and $\theta(\mathbf{s})$ performs element-wise non-linear transformation.

$$\hat{\mathbf{y}} = \theta(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2.5}$$

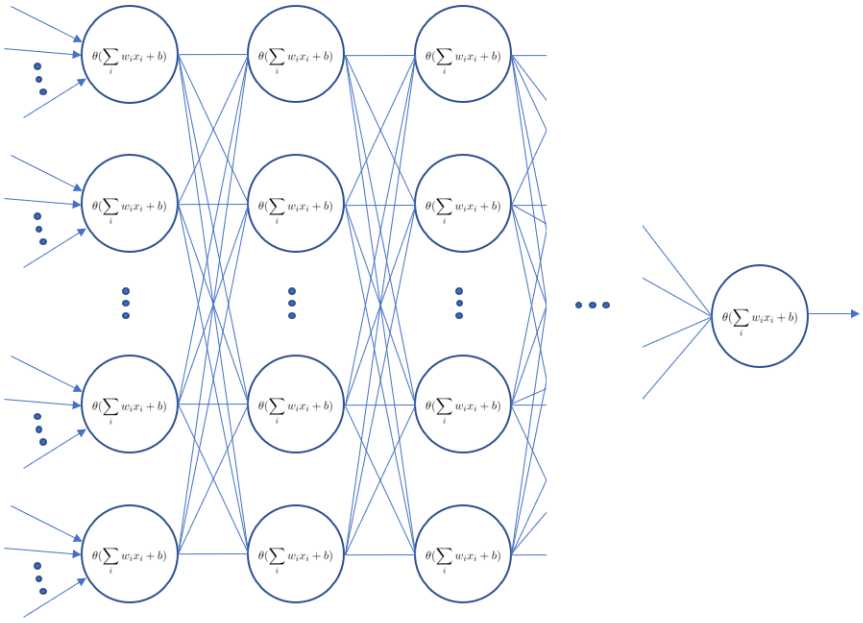


Figure 2.5: Fully-Connected Neural Network

2.2.2 Recurrent Neural Networks

Traditional neural networks have found success in many fields but it is unable to capture temporal dependencies in the data, making it unsuitable for signal processing applications. Recurrent neural network on the other hand was developed specifically for time-varying data, which contains additional memory states that retain and process information from previous time steps. Figure 2.6 below shows a RNN unit.

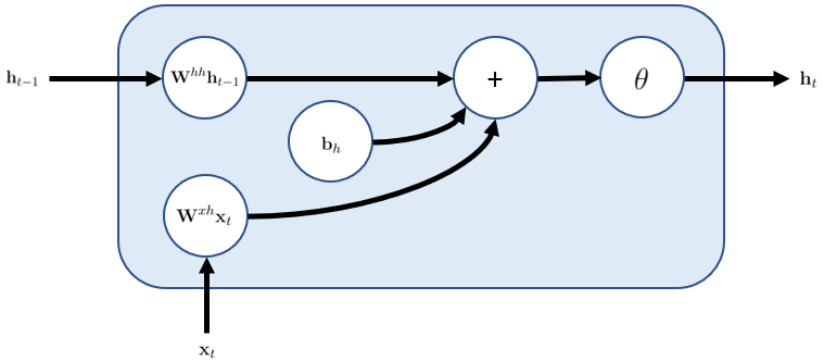


Figure 2.6: RNN Unit

A standard RNN unit takes as input a feature vector $\mathbf{x}_t$ at time t and the hidden state $\mathbf{h}_{t-1}$ from the previous time step (which contains information from the time steps before it), computes the product with weight matrices $\mathbf{W}^{xh}$ and $\mathbf{W}^{hh}$, adds bias vector $\mathbf{b}^h$ and applies a non-linear activation (usually *tanh*) to produce the current time-step's hidden state $\mathbf{h}_t$. Finally, the output $\mathbf{h}_t$ is in turn propagated to the next time step.

Mathematically, the operation that this unit carries out is defined in equation 2.6, where the objects in the equation have dimensions $\mathbf{h}_t \in \mathbb{R}^N$, $\mathbf{W}^{hh} \in \mathbb{R}^{N\text{x}N}$, $\mathbf{W}^{xh} \in \mathbb{R}^{N\text{x}M}$, $\mathbf{b}^h \in \mathbb{R}^N$. In addition, the weight matrices superscript letters indicates the mapping (*hh*: from hidden states to hidden states, *xh*: from input features to hidden states).

$$\mathbf{h}_t = \theta(\mathbf{W}^{hh}\mathbf{h}_{t-1} + \mathbf{W}^{xh}\mathbf{x}_t + \mathbf{b}^h) \tag{2.6}$$

The RNN unit uses hidden states propagated through units that store information through time. Therefore, the training of this particular network via back-propagation has the special meaning of backward stepping through time, which can capture temporal properties of a signal such as auto-regression and moving average.

2.2.3 Activation Functions

Activation functions transform the output of a neural network unit element-wise, allowing it to model non-linear functions. For this project, only the sigmoid, softmax and tanh activation functions are considered, which are defined in equations 2.7, 2.8 and 2.9 respectively, where $\mathbf{y}[i]$ is a scalar element of an array.

$$f_{sigmoid}(\mathbf{y}[i]) = \frac{1}{1 + e^{-\mathbf{y}[i]}} \tag{2.7}$$

$$f_{softmax}(\mathbf{y}[i]) = \frac{e^{\mathbf{y}[i]}}{\sum_i e^{\mathbf{y}[i]}} \tag{2.8}$$

$$f_{tanh}(\mathbf{y}[i]) = \frac{2}{1 + e^{-2\mathbf{y}[i]}} - 1 \tag{2.9}$$

The sigmoid function compresses the input in a range between 0 and 1 and can be used to approximate virtually any decision boundary with high enough number of units [23]. The softmax function compresses the range between 0 and 1 as well, but the output sums up to 1 and hence can be interpreted as probabilities. The tanh function is often used in the recurrent layer and can take both positive and negative values.

2.2.4 Loss Functions

Loss functions are objective functions that the neural network aims to minimize when being trained. For this project, the loss function of concern is the cross-entropy loss defined in equation 2.10, where $\mathbf{y}$ is an array that contains the true distribution of N possible categories in which only one category is 1 and all others are 0, whereas $\hat{\mathbf{y}}$ is the estimated probability distribution. This particular function penalizes differences in probability distributions and is suitable for multi-class classification problems.

$$l_{crossentropy}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{N} \mathbf{y}[i] log(\hat{\mathbf{y}}[i]) \tag{2.10}$$

2.2.5 Neural Network Training

As any other machine learning algorithm, neural networks aim to learn an underlying pattern present in the data by minimizing an error measure defined by a loss function given some sample data (a process known as training). Formally, this problem is done by finding the optimal weights of the model per layer as defined in equation 2.11, where $\hat{\mathbf{y}}$ is the estimate of the true label $\mathbf{y}$ and l is the loss function.

$$\mathbf{W}_{opt} = \text{argmin}_{\mathbf{W}}\{\mathbb{E}(l(\hat{\mathbf{y}}, \mathbf{y}))\} \tag{2.11}$$

This optimization problem is often carried out with stochastic gradient descent (SGD) [24] which is an iterative optimization method that changes the parameters (weight matrices) of the network. The update diminishes the target loss function iteratively at k^{th} step by updating the weights as $\mathbf{w}_k = \mathbf{w}_{k-1} - \gamma \frac{\nabla \hat{R}(\mathbf{w})}{||\nabla \hat{R}(\mathbf{w})||}$ where $\hat{R}(\mathbf{w}) = \mathbb{E}[l(\hat{\mathbf{y}}, \mathbf{y})]$ is the average loss incurred using the training data batch and γ is a hyper-parameter that controls the speed of learning (learning rate).

To apply the the SGD across multiple layers, the back-propagation procedure is used [25], which consists of initializing all weights randomly, iteratively compute the gradient of the loss function with respect to last layer's output and proceeds sequentially from last to first layer. The learning is therefore a real life application of differential chain rule and as such, the choice of loss functions, activation functions and forward computation must be differentiable for the learning to occur.

Note that other optimization methods such as ADAM and RMSProp exist, which have different properties and perform better in certain cases (such as problems with sparse gradients). However, the rest of the paper focuses on the SGD method in particular which is the most well understood and is applicable to the TT format [17].

2.3 Financial Terminologies

2.3.1 Asset Classes

The financial markets produce natural tensor data across a range of assets, which can be divided into four major categories: equities, currencies, commodities and fixed income assets.

Equities represent ownership in a company and the asset value is mostly tied to the underlying company's future cash flow prospects. Currencies are traded in pairs and their value is mostly tied to the economic outlook of given countries. Commodities are natural resources which include metals, agricultural products and energy resources. Finally, fixed income assets are loans that provide fixed payments over time and can be issued by the government [26].

The asset classes are governed by complex relationships and can impact one another in several ways. For instance, increase in US interest rates often increases the value of US dollars (USD) since it attracts foreign investments. Rising USD can impact the trading of major commodities since most commodities are USD denominated. This in turn can impact companies that rely on the import of those commodities and hence influencing the equities market [27].

Overall, the financial markets produces market data such as pricing and volume information across a range of asset classes which exhibit structural relationships, hence making it ideal for tensor processing methods.

2.3.2 Financial Instruments and Indices

There exist a range of financial instruments called derivatives whose value derive from the underlying asset. An example of this are futures contracts, which are agreements to buy or sell an asset at an agreed price at a given time. These contracts are often used for hedging purposes to protect an investor from an unfavourable change in price. For commodities such as gold, futures contracts can be used as an indicator for the amount of uncertainty in the economy since gold is a safe haven asset that rises in value during economic downturns [28].

In addition, there exist a number of indices that provide information on the underlying asset class. Stock market indices for example monitor the major stocks of a given country and can be used to assess the overall economic activity. In addition, bond yields track the returns that an investor can make on bonds, which monitors the demand of fixed income market and can be used as a proxy for economic uncertainty since bonds are more in demand during bear markets.

Chapter 3

Tensor-Train Neural Networks

3.1 Mathematical Derivation

3.1.1 Tensor-Train Fully-Connected Layer

Multiple papers [17] [5] have shown that by applying the TTD to the fully-connected layer, it is possible to drastically reduce the dense weight matrix while preserving its expressive power, hence reducing the complexity characteristics of neural networks with high-dimensional data. The main idea behind is that the fully-connected layer $\hat{\mathbf{y}} = \mathbf{W}\mathbf{x} + \mathbf{b}$ can be written in the tensor format by reshaping the input vector $\mathbf{x} \in \mathbb{R}^M$, output vector $\hat{\mathbf{y}} \in \mathbb{R}^N$ and the weight matrix $\mathbf{W} \in \mathbb{R}^{N \text{x} M}$ into multi-modal tensors by establishing a bijection (one-to-one mapping) via the reshaping operation.

To derive the TT fully-connected layer (TTFCL), it is important to first express the tensor train format mentioned in the previous section with double indices. For instance, given a mode-d tensor $\mathcal{A} \in \mathbb{R}^{p_1 \text{x} p_2 \text{x} \ldots \text{x} p_d}$, if each integer p_k can be factorized as $p_k = m_k n_k$, then each of the TT cores can be expressed with 2 indices i_k and j_k as $\mathcal{G}_k[l_k] = \mathcal{G}_k[i_k, j_k]$. This allows the TT format to be re-written as equation 3.1.

$$\mathcal{A}[l_1, ..., l_d] = \mathcal{A}[(i_1, j_1), ..., (i_d, j_d)] = \mathcal{G}_1[i_1, j_1] ... \mathcal{G}_d[i_d, j_d] \tag{3.1}$$

The fully connected layer equation can be written in the index form as shown in equation 3.2. If both M and N can be factorized into a series of d integers m_i and n_i for $i = 1, ..., d$ respectively (meaning that $\prod_{k=1}^{d} m_k = M$ and $\prod_{k=1}^{d} n_k = N$), the weight matrix, the input vector and the output vector can be reshaped into order-d tensors and the index form equation can be re-written as equation 3.3, converting it into a tensor operation.

$$\hat{\mathbf{y}}[j] = \sum_{i=1}^{M} \mathbf{W}[i, j] \cdot \mathbf{x}[i] + \mathbf{b}[j] \tag{3.2}$$

$$\hat{\mathcal{Y}}[j_1, ..., j_d] = \sum_{i_1=1}^{m_1} ... \sum_{i_d=1}^{m_d} \mathcal{W}[(i_1, j_1), ..., (i_d, j_d)] \cdot \mathcal{X}[i_1, .., i_d] + \mathcal{B}[j_1, ..., j_d] \tag{3.3}$$

Given the above format, it is easy to observe the equivalence between the weight tensor $\mathcal{W}$ in equation 3.3 and the tensor $\mathcal{A}$ in equation 3.1. Therefore, given an input tensor $\mathcal{X}$ (or any input data that can be reshaped as tensors), it is possible to describe the fully connected layer as a tensor operation, where the weight matrix $\mathbf{W}$ can be effectively represented as a tensor $\mathcal{W}$ which can be stored in the TT format as shown in equation 3.4 (the weight tensor stored in TT format will be referred to as the TT-matrix for the remaining of the paper).

$$\hat{\mathcal{Y}}[j_1, ..., j_d] = \sum_{i_1=1}^{m_1} ... \sum_{i_d=1}^{m_d} \mathcal{G}_1[i_1, j_1]...\mathcal{G}_d[i_d, j_d] \cdot \mathcal{X}[i_1, .., i_d] + \mathcal{B}[j_1, ..., j_d] \tag{3.4}$$

To recap, $\mathcal{X}$ is an order-d tensor with shape $(m_1, ..., m_d)$, $\mathcal{Y}$ and $\mathcal{B}$ are order-d tensors with shape $(n_1, ..., n_d)$, $\mathcal{W}$ is an order-$2d$ tensor with shape $(m_1, n_1, ..., m_d, n_d)$ and $\mathcal{G}_k$ are order-4 tensors with shapes (m_k, n_k, r_{k-1}, r_k) for $k = 1, ..., d$. This particular tensor-train format with order-4 core tensors is also known as the matrix product operator (MPO) [1].

Therefore, to control the number of parameters in a layer, it is possible to vary the number of hidden units ($\prod_{k=1}^{d} n_k = N$) as well as the TT-ranks (hence the number of cores slices). By keeping the TT-ranks small, it is possible to achieve super-compression as detailed in section 2.1.4. Specifically, the double indexing format allows the storage of $\sum_{k=1}^{d} m_k n_k r_{k-1} r_k$ elements

(number of parameters of all d cores) instead of $\prod_{k=1}^{d} m_k n_k = MN$ (number of parameters in the fully connected layer), which is more memory efficient if the ranks are small.

The hyper-parameters of the TT format will influence not only the complexity of the network, but also the performance by controlling the number of hidden units. The improvement in complexity and performance will be a trade-off since the tensor train method is a compression method that only approximately reconstruct the original weight matrix.

3.1.2 Tensor-Train Recurrent Layer

Despite being more complex than the classical fully-connected layer, the recurrent layer is fundamentally composed of multiple fully-connected layers of the form $\hat{\mathbf{y}} = \theta(\mathbf{W}\mathbf{x}+\mathbf{b})$ which are jointly trained over multiple time steps. Therefore, the derivation of the TT-RNN architecture is relatively straight forward, since all that needs to be done is to replace the classical feed-forward operations with the TTFCL equivalent. Therefore, the computation of the hidden state of the classical RNN discussed in section 2.2.2 can be formulated as in equation 3.5 where $\mathrm{TT}(\mathbf{W}^{xh}\mathbf{x}_t)$ denotes the TT-matrix by explicit vector multiplication.

$$\mathbf{h}_t = \theta(\mathbf{W}^{hh}\mathbf{h}_{t-1} + \mathrm{TT}(\mathbf{W}^{xh}\mathbf{x}_t) + \mathbf{b}_h) \tag{3.5}$$

3.2 Algorithmic Implementation

3.2.1 Matricized Computation

It is possible to implement the tensor-train layer with modern neural network libraries such as Keras [29] by the direct application of equation 3.4 like in papers [17] [5]. The algorithmic implementation of this equation is done by a series of matrix products between the matricized/reshaped input tensor and tensor-train cores. This algorithm is originally developed by A. Novikov [17] and is described below, where the mathematical objects, mathematical operations and algorithmic operations are colored in purple, gray and black respectively.

The algorithm computes the forward-pass of the tensor-train layer with an order-d input tensor $\mathcal{X} \in \mathbb{R}^{m_1 \text{x} ... \text{x} m_d}$ and an order-$2d$ weight tensor $\mathcal{W} \in \mathbb{R}^{(m_1 \text{x} n_1) \text{x} ... \text{x} (m_d \text{x} n_d)}$ stored in tensor-train format with order-4 cores $\mathcal{G}_k \in \mathbb{R}^{m_k \text{x} n_k \text{x} r_{k-1} \text{x} r_k}$ for $k = 1, 2, .., d$. The result of this computation is an order-d output tensor $\mathcal{Y} \in \mathbb{R}^{n_1 \text{x} ... \text{x} n_d}$.

To run the algorithm, it is necessary to have a pre-defined array that specifies the shape of the input tensor $\mathbf{m} = [m_1, m_2, ..., m_d]$, the shape of the output tensor $\mathbf{n} = [n_1, n_2, ..., n_d]$, the tensor-train ranks $\mathbf{r} = [1, r_1, ..., r_{d-1}, 1]$ and a collection of cores $\mathcal{G} = [\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_d]$. It is important to note that all the arrays below have indices that start with 1, which means that $\mathbf{r}[k] = r_{k-1}$, $\mathbf{m}[k] = m_k$, $\mathbf{n}[k] = n_k$ and $\mathcal{G}[k] = \mathcal{G}_k$.

Algorithm 1 Tensor-Train Fully-Connected Layer Forward Pass

```
1:  procedure ForwardPass (𝒳,𝒢,m,n,r)
2:      M ← 𝒳.reshape(−1, m[d]r[d + 1])
3:      for k = d to k = 1 update k←k−1 do
4:          M ← M.reshape(−1, m[k]r[k + 1])
5:          G ← 𝒢[k].reshape(m[k]r[k + 1], n[k]r[k])
6:          M ← M G
7:          M ← M.reshape(-1, n[k])
8:          M ← M^T
9:      end for
10:     𝒴 ← M.reshape(n[1], ..., n[d])
11:     return 𝒴
12: end procedure
```

3.2.2 Interpretation

This section explores the above implementation and draws links between the TT cores and the modes of the original input tensor. For that goal, the above algorithm is broken down and visualized graphically for the case of an order-3 input tensor $\mathcal{X}$.

Firstly, it is important to visualize the weight matrix cores $\mathcal{G}_k \in \mathbb{R}^{m_k \text{x} n_k \text{x} r_{k-1} \text{x} r_k}$ reshaped as matrices $\mathbf{G}_k$ of size $(m_k r_k, n_k r_{k-1})$ for $k = 1, 2, .., d$ as an re-organization of the tensor-train core slices. This is shown in figure 3.1 where each (i_k, j_k) pair points to a core slice of shape (r_k, r_{k-1}) where $i_k = 1, 2, ..., m_k$ and $j_k = 1, 2, ..., n_k$. As special cases, since $r_0 = r_d = 1$ for $k = 1$ and $k = d$, (i_k, j_k) pairs points to fibers instead of slices.

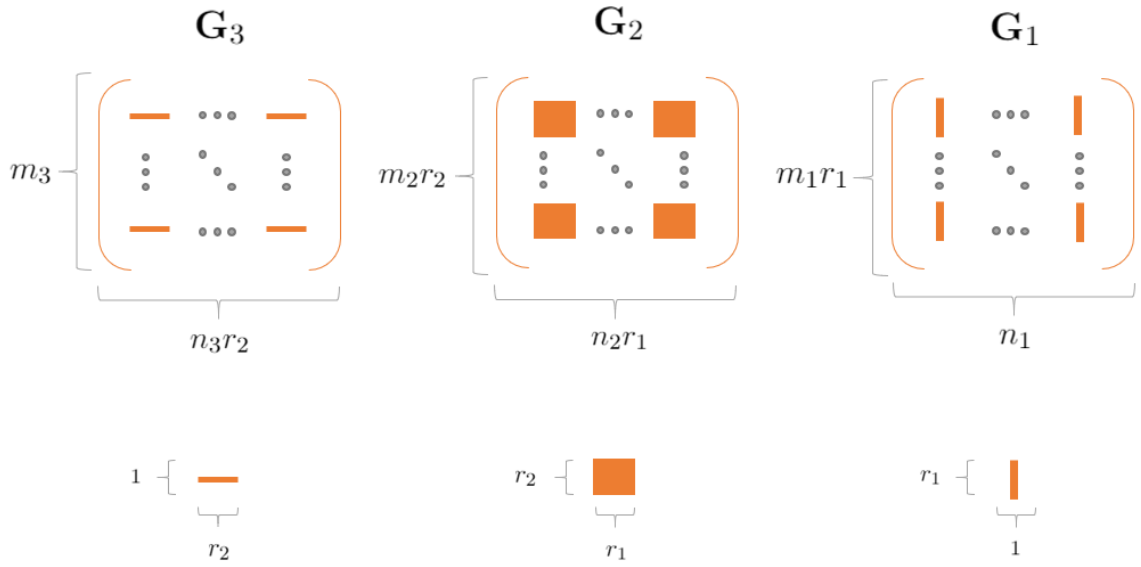


Figure 3.1: Cores Visualization

For the first step of the algorithm ($k = d = 3$), the input tensor $\mathcal{X}$ is reshaped into the transpose of the mode-3 unfolded matrix. The new matrix $\mathbf{X} \in \mathbb{R}^{m_1 m_2 \text{x} m_3}$ is therefore composed of $m_1 m_2$ rows where each row is a mode-3 fiber of the original input tensor $\mathcal{X}$ (the rows can be enumerated as $\mathcal{X}[1, 1, :], \mathcal{X}[1, 2, :], ..., \mathcal{X}[1, m_2, :], \mathcal{X}[2, 1, :], \mathcal{X}[2, 2, :], ..., \mathcal{X}[m_1, m_2, :]$) [30].

For example, if 3 modes of the input tensor corresponds to time, features and assets, then each row of the reshaped matrix corresponds to one feature at a one given point in time across all assets. Specifically, the first m_2 rows corresponds to all m_2 features at one given point in time

across all assets, while the next m_2 rows corresponds to the same features but at the successive point in time. Therefore, each column contains information relating to features over time for 1 asset.

Once reshaped, a new matrix $\mathbf{M}$ of shape $(m_1 m_2, n_3 r_2)$ is computed by performing the product on $\mathbf{X}$ and $\mathbf{G}_3$, which is illustrated in the figure below.

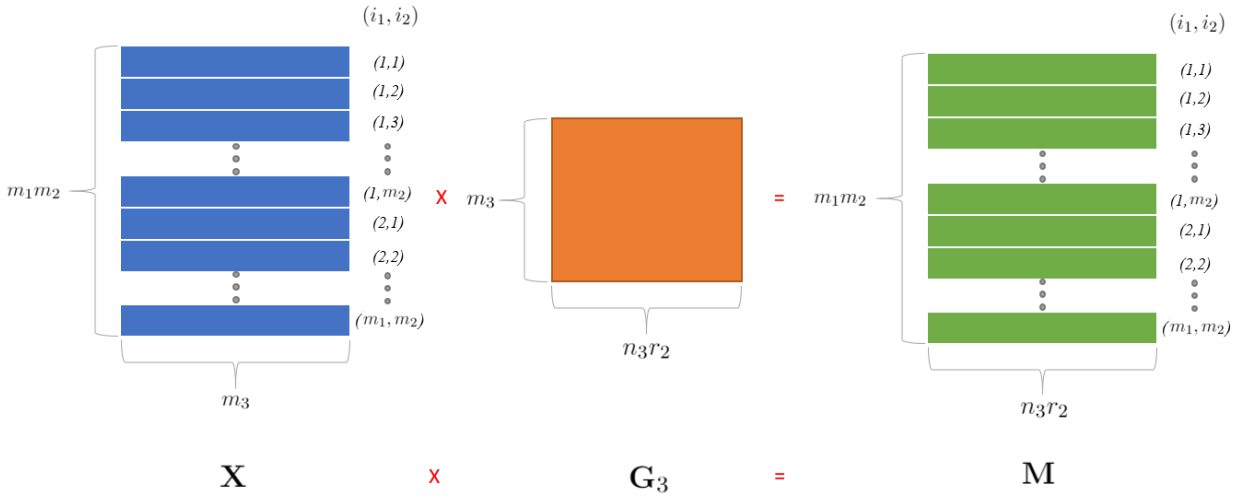


Figure 3.2: Input-Core Product Visualization ($k = d = 3$)

This particular product is important since it can be interpreted using the bases view as a linear combination of the mode-3 bases of the original tensor data and the factors from the TT cores, hence establishing links between the data structure and the processing. For instance, $\mathbf{X}$ can be seen as a composition of column vectors (bases), while $\mathbf{G}_3$ can be seen as a composition of row vectors (factors). The matrix product can therefore be interpreted as a sum of outer products of bases and factors, which contains information relating to mode-3 aspect of the data [31]. This is shown in figure 3.3 below.

$$\mathbf{X} = |\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_{m_3}| \tag{3.6}$$

$$\mathbf{G}_3 = |\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_{m_3}|^T \tag{3.7}$$

$$\mathbf{X}\mathbf{G}_3 = \mathbf{b}_1 \circ \mathbf{f}_1^T + \mathbf{b}_2 \circ \mathbf{f}_2^T + ... + \mathbf{b}_{m_3} \circ \mathbf{f}_{m_3}^T \tag{3.8}$$

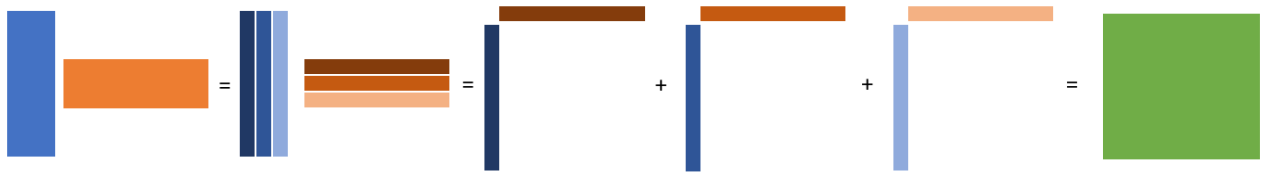


Figure 3.3: Bases View

Using the analogy from before, each of the bases of the input matrix (blue columns) corresponds to data relating to 1 asset only. Therefore, each vector-row outer product forms a matrix relating to 1 specific asset, while the overall matrix product is the sum of these matrices.

An important aspect to keep in mind is that each row of resulting matrix $\mathbf{M}$ (the green matrix in figure 3.2) still retains information that is fundamentally related to mode-3 fibers of the original tensor. Each row can still be enumerated with the pairs $(i_1, i_2) = (1,1), (1,2), ..., (m_1, m_2)$. Subsequently, when the matrix $\mathbf{M}$ is reshaped from shape $(m_1 m_2, n_3 r_2)$ to shape $(m_1 m_2 r_2, n_3)$, every r_2 rows will correspond to 1 pair of (i_1, i_2) exactly (figure 3.4). This information is still retained column wise when the matrix is transposed.

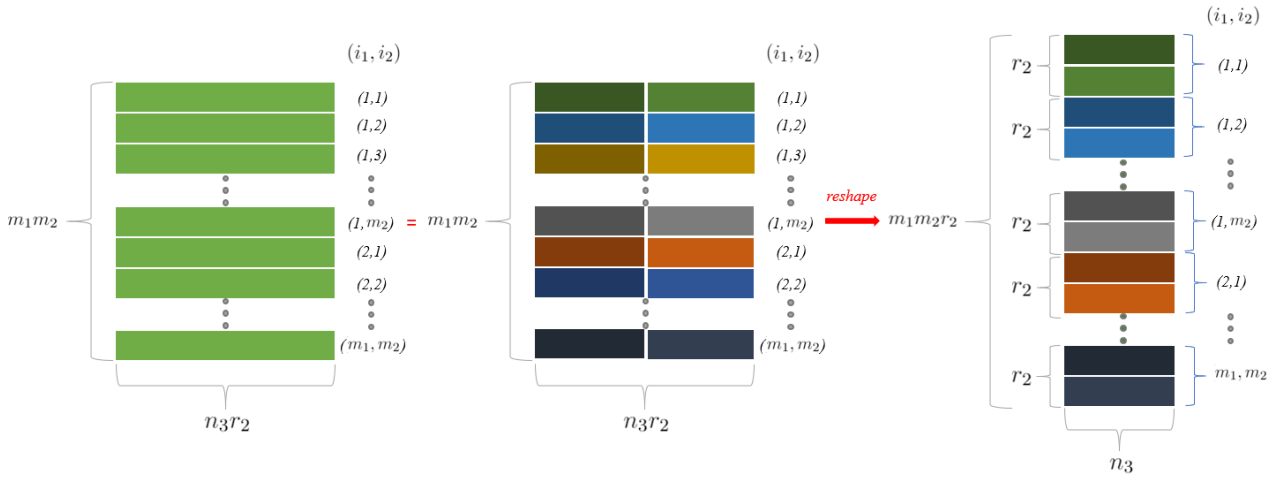


Figure 3.4: $\mathbf{M}$ Reshape Visualization ($k = d = 3$)

For the second iteration of the for loop, the matrix M is then reshaped again from $(n_3, m_1 m_2 r_2)$ to $(n_3 m_1, m_2 r_2)$ which repeats the entire logic mentioned in this section. An important aspect to note is that since the reshaping operation is sampling the rows of the new matrix every complete enumeration of $i_2 = 1, ..., m_2$ (see figure 3.5), the new columns of the reshaped matrix corresponds to mode-2 information of the data, and the rows of the core $\mathbf{G}_2$ can be interpreted as factors corresponding to mode-2 and so on.

Therefore, using the analogy from before, each column contains information relating to 1 specific feature over assets and time, while the matrix multiplication with $\mathbf{G}_2$ combines this information on a per-feature basis.

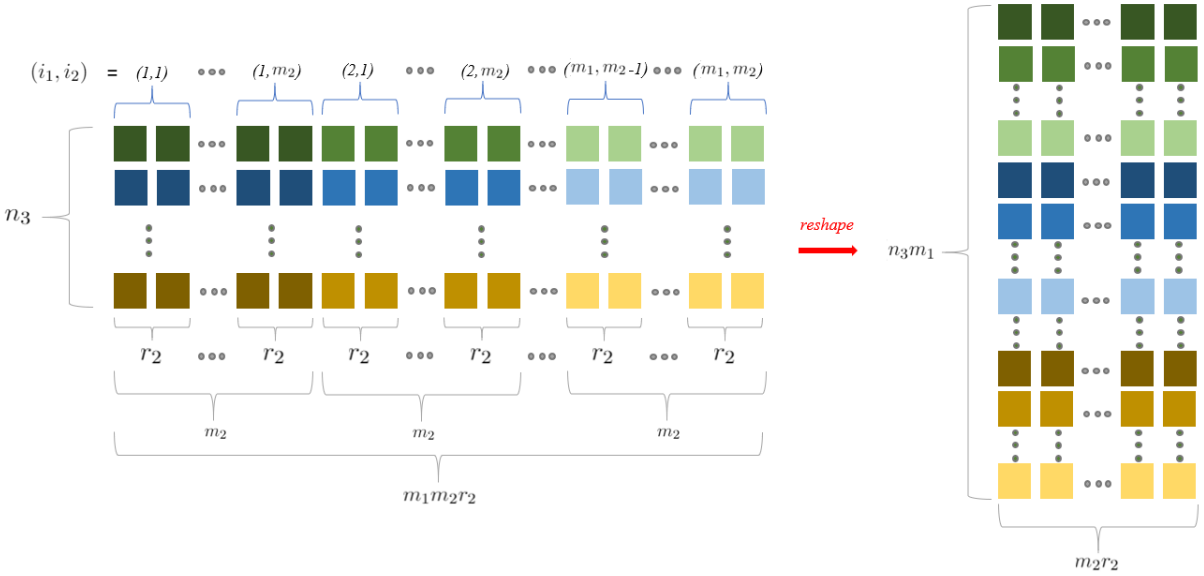


Figure 3.5: $\mathbf{M}$ Reshape Visualization ($k = d - 1 = 2$)

Finally, the algorithm will produce a final matrix of shape $(n_1, n_2 n_3)$ at the end of the for loop, which can be reshaped according to $\mathbf{n}$ as (n_1, n_2, n_3) which is the output tensor $\mathcal{Y}$.

3.3 Keras Implementation

3.3.1 TTFCL and TTRL Classes

The implementation of the TT-Layers is done using python 3 with the Keras deep learning framework [29]. The Keras framework allows the creation of layer classes that is compatible with multiple deep learning operations.

Firstly, the TT-Fully Connected Layer (TTFCL) class is initialized by taking as input the input tensor shapes $m_1, ..., m_d$, the output tensor shapes $n_1, ..., n_d$ and the TT-ranks $r_0, ..., r_d$. Using the inputs, a total of $\sum_{k=1}^{d} m_k n_k r_{k-1} r_k$ parameters are initialized randomly as the TT-weights.

Once the weights are initialized, the total number of parameters are partitioned into their respective cores, where each core contains $m_k r_k n_k r_{k-1}$ parameters. The partition is recorded using an array of pointers which point to where each core's parameters begin and end. These cores are then reshaped as $(m_k r_k, n_k r_{k-1})$ and used for the forward computation, which is a direct implementation of the algorithm outlined in section 3.2.1.

Finally, the same exact procedure is carried out for the feature mapping matrix of the recurrent neural network. Please see appendix C for the python implementation of these layers.

3.3.2 Training and Complexity

For the training of the TT-layers, it is possible to compute the gradient of the loss function w.r.t. the original weight matrix, convert to TT format and update iteratively by composing and decomposing the weight tensor. However, this is highly inefficient in terms of computation. A better method is to update the weights using the back-propagation algorithm and apply an optimizer (such as SGD) directly w.r.t. to the TT-cores. This allows the TT layer to be jointly trained with other hidden layers in the architecture (see paper [17] for more details). For Keras, the layer class computes the backward pass and applies the backpropagation algorithm directly on the initialized trainable weights, which is compatible with SGD.

In the paper *Tensorizing Neural Networks* [17], authors studied the complexity characteristics between the classical fully-connected layer and its TT based counterpart. The complexities are summarized in the table below, where d is the number of cores, $m = \max(m_k)$ for $k = 1, 2, ..., d$, $r = \max(r_k)$ for $k = 0, 2, ..., d$, M and N are the number of columns and rows of the original weight matrix respectively.

Operation	Time	Memory
FC forward pass	$O(MN)$	$O(MN)$
TT forward pass	$O(dr^2m \max(M, N))$	$O(r \max(M, N))$
FC backward pass	$O(MN)$	$O(MN)$
TT backward pass	$O(d^2r^4m \max(M, N))$	$O(r^3 \max(M, N))$

Table 3.1: TT Layer Complexity

3.4 Optimal Tensor Train Networks Design

To design an optimal TT neural network architecture, it is necessary to follow a few rules. Firstly, the input tensor should be reshaped such that it has balanced values for each of the mode. If the mode values are too small or if there are too few modes, the performance tends to be worse [17].

Secondly, the input tensor should be reshaped to the highest order possible without losing physical significance. For instance, most of the compression power comes from reshaping the input tensor into a higher order tensor using low TT-ranks. However, as it is shown in section 3.2.2, the cores of TT train are interpretable only if the input tensor have modes that match physical properties of the data. Therefore, excessive reshaping/quantization should be avoided.

Finally, if the TT-layer is used as the last layer, the penultimate TT-rank r_{d-1} (known as the bottleneck rank) [6] should be at least equal to the dimensionality of the output. This means that for a 10-classes classification problem with one-hot encoding, r_{d-1} should be at least 10.

Chapter 4

Financial Application

4.1 Data Selection

Before diving in the application, it is important to specify the end goal, select a universe of assets that is sensible in the financial sense and format the data as a tensor.

The end goal of this financial application is to forecast the next day returns of the Japanese Yen against the United States Dollars (JPY/USD). This foreign exchange (FX) pair relates the world's biggest economy's currency USD against JPY, which is often considered as a safe-haven currency (a currency that rises in value during turbulent economic circumstances) [28]. Therefore, this pair reflects the global economic outlook in terms of bullish-vs-bearish.

The universe of assets taken into consideration is chosen to reflect the end goal of this particular application, which is summarized in table below. Overall, the assets are chosen to reflect several asset classes (equities, currency pairs, commodities and fixed income) and type (assets, indices and derivatives). A diverse set of assets is important due to the interconnected world economies and their relative relationships. For instance, the rising price of certain commodities such oil can impact the economic output of a country that is highly dependent on the export/import of that commodity, which can in turn affect its currency value [32]. It is therefore important to capture as many interconnections as possible.

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

Table 4.1: Financial Data Selction

In terms of specific asset choices, the assets are chosen to reflect US and Japan's biggest trading partners (e.g. Canada and China) as well as world's biggest economies from different continents. In terms of business activities, this is captured by the stock market index of various countries. In terms of foreign exchange, this is reflected in each country's official currency. In terms of fixed income assets, this is reflected in the 10-years government yield as it reflects the relationship between the demand of bonds and interest rates [33]. In addition, major commodities futures such as oil and copper are also taken into consideration since each country relies on the export/import of these materials to different degrees. Finally, special assets include gold and Swiss Franc which often act as safe-haven assets.

4.2 Data Pre-Processing

4.2.1 Raw Data Formatting

The data of the above assets were obtained from the Bloomberg Terminal [34] and contain the daily opening, high, low and closing values (O/H/L/C) from January 2006 to May 2019. For the stock market indices, trading volume is included (V) which signals trading activity. Finally, commodity futures data include also open interests (OI) which is highly correlated with volume and signals aggregate interests in the future market. An example of the raw data is presented in figure 4.1, which shows the time-series of commodity futures (see appendix in section B for graphs of all assets).

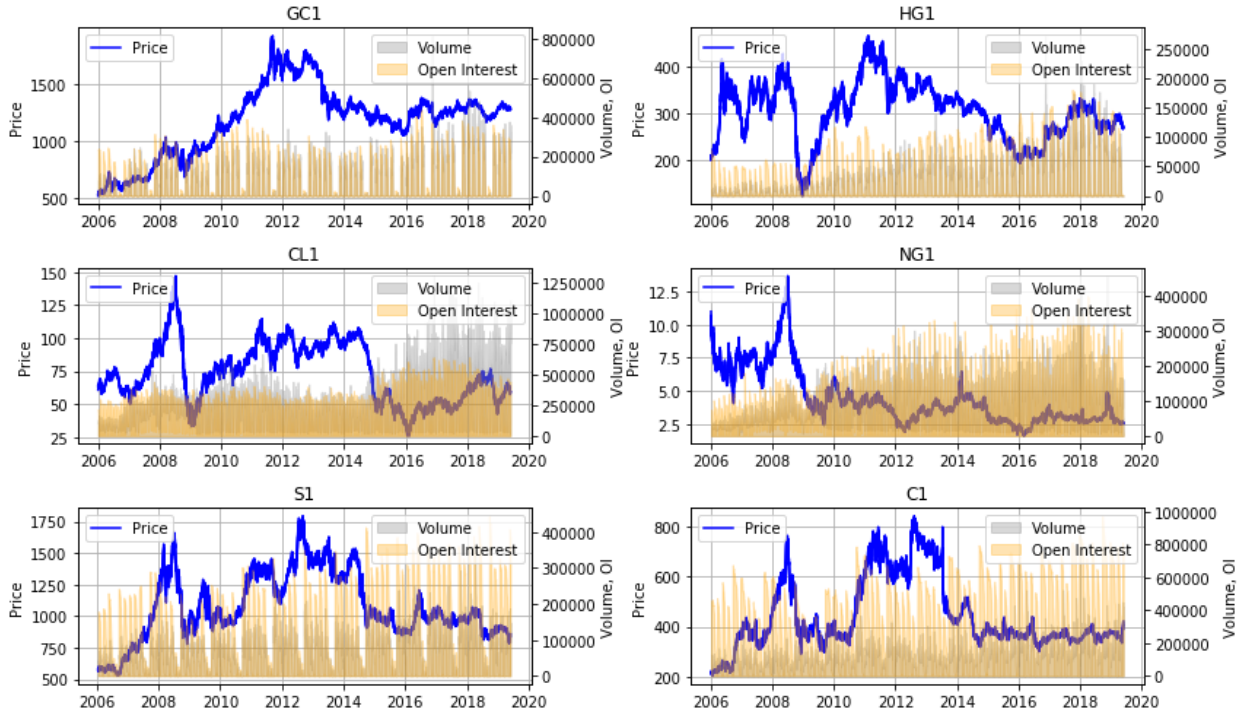


Figure 4.1: Commodity Futures

Finally, to reformat the data-set as a tensor, columns of zeros are added to assets which lack one of the O/H/L/C/V/OI values such that all assets have 6 raw market data features (these values will be referred to as the raw features for the remaining of the report). Therefore, the dataset is now a 4th order natural tensor $\mathcal{D}_{raw} \in \mathbb{R}^{3000\text{x}4\text{x}6\text{x}4}$ with the modes corresponding to time steps, raw features, assets per asset class and asset classes.

4.2.2 Stationarity and Memory

Neural network models work the best when the input is stationary and bounded. A stochastic signal is considered wide-sense stationary (WSS) if the first moment (the mean) is time invariant and the autocorrelation function is dependent on time differences only.

$$m_x(t) = m_x(t+T), \text{for any } T \tag{4.1}$$

$$R_{xx}(t_1, t_2) = R_{xx}(t_1 - t_2), \text{for any } t_1, t_2 \tag{4.2}$$

Stationarity is an important characteristic to explore in financial markets, since such characteristics would imply that the signal's average persists over time and that the auto-correlation is dependent on time differences only. This would justify investing strategies based on mean-reverting behaviour of the signal. However, for most assets, the price doesn't exhibit this behaviour since the world economy tends to rise over time. This can be seen in figure 4.2 where most asset prices rise without bound.

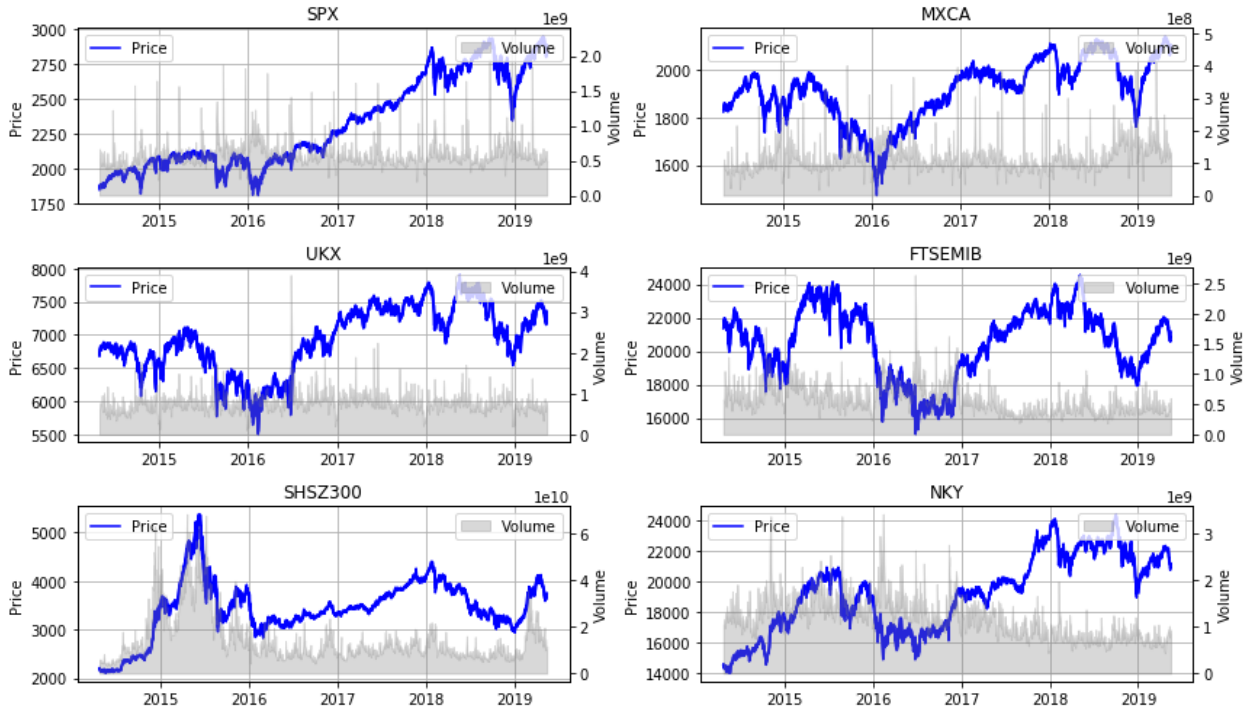


Figure 4.2: Market Indices

In terms of machine learning, non-stationarity often means that the underlying process that generates the data changes, which implies a change in data distribution that can worsen the performance of the model. In the financial sense, the change in underlying process can indicate various market regimes that could be caused by change in government policies, trade wars, etc.

The increasing/unbounded value of assets is good for the economy but bad for modelling. For instance, data scaling is often used to pre-process the data to compress the range within the non-linear regions of activation functions for optimal performance, which is not possible if the values can increase without boundary.

In quantitative finance, a classical approach is to transform the pricing data through differencing the log-transform of prices, which is known as log-returns. This is because the returns are assumed to be log-normally distributed in the short-run (in the long run, the upward trend of the market tends to skew the distribution). The equations below define the simple and log-returns where p_t denotes the price of a given asset at time t.

$$r_t^{simple} = \frac{p_t - p_{t-1}}{p_{t-1}} \tag{4.3}$$

$$r_t^{log} = ln(\frac{p_t}{p_{t-1}}) = ln(p_t) - ln(p_{t-1}) \tag{4.4}$$

However, the major draw-back of using returns is that the data loses memory when differencing the log-prices [35], which is not the case when prices are used directly since the price level reflects how much the market has valued a given asset up to a certain point in time. A method of solving this stationarity vs memory dilemma is to introduce additional stationary features that contain memory of past price values, such as moving statistics and relative price levels with respect to past values.

4.2.3 Features Generation

The raw price levels do not follow a Gaussian distribution for most asset prices and are unbounded, while the volume (and open interest) follow a nicer distribution. The neural network model can work with volume and open interests directly, but the prices must be first processed as discussed in section 4.2.2. Therefore, the first feature needed in the input data is the log-return. This is shown in figure 4.3, where the log-returns and the raw-volume follow a nicer distribution than the original FTSE MIB prices.

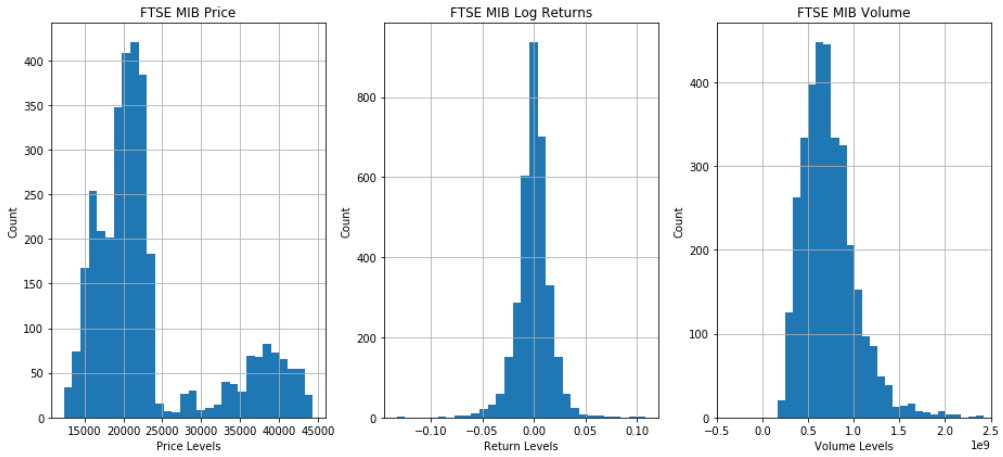


Figure 4.3: Features Histograms FTSE MIB

Additional features are manufactured to introduce memory such as rolling statistics of the log-returns. These include the rolling average, standard deviation, skew and kurtosis, which are computed on a weekly, bi-weekly and monthly basis (rolling window of 5, 10, 22). Additional non-statistical descriptive features are also created to incorporate information of past prices such as the RHLC (Relative High-Low-Close) feature, the RMM (Relative Minimum-Maximum) feature and the HLS (High-Low Spread) feature which are defined in the equations below, where $P_C(t), P_H(t), P_L(t), T$ denote the closing, high, low prices at a given day t and the rolling window period respectively.

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

$$RMM(t) = \frac{P_C(t) - \min P_L(t - T : t)}{\max P_H(t - T : t) - \min P_L(t - T : t)} \tag{4.6}$$

$$HLS(t) = \frac{P_H(t) - P_L(t)}{P_L(t)} \tag{4.7}$$

Figure 4.4 below shows the distribution of new features for NKY (using only the rolling window of 22 where appropriate). The rolling statistics follow nicer distributions, while the price features are bounded between 0 and 1. The dataset is now a 4th order tensor $\mathcal{D} \in \mathbb{R}^{3000\text{x}20\text{x}6\text{x}4}$ with the modes corresponding to time steps, new features, assets per asset class and asset classes.

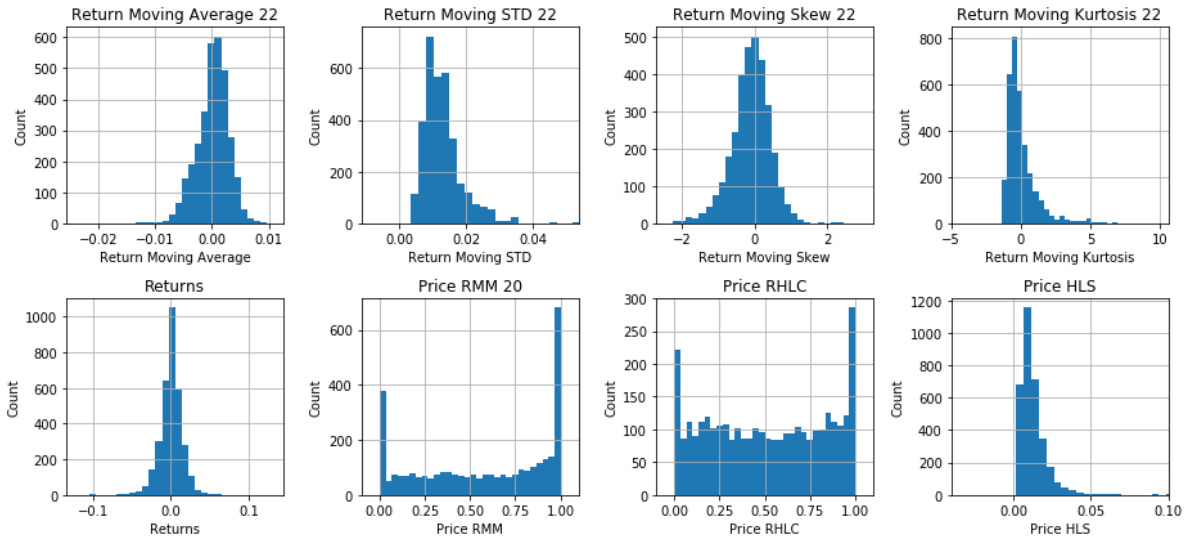


Figure 4.4: New Features Histograms NKY

4.3 Labelling and Performance Evaluation

The labels needed for the supervised machine learning problem is simply the original log-returns of the JPY/USD series shifted back in time by 1 time step. This way, the models are learning to forecast the next day returns at time $t + 1$ as a function of past features at time t. The supervised learning problem is further defined as a classification problem where negative, neutral and positive returns are labelled as {0,1,2} which is further encoded with one-hot encoding as {100, 010 and 001} respectively.

In terms of performance evaluation, the model is assessed in three aspects. Firstly, the modelling performance is measured using the loss function (categorical cross entropy) and the accuracy (percentage) of the prediction. Secondly, the modelling complexity is measured as the number of trainable parameters needed in the TT layer and the time needed to complete the training. Finally, the trading performance is measured using the metrics defined in the table below.

Trading Performance Metric	Description	Preference
Total Return (%)	Final total log-return	High
Winning Trades (%)	Percentage of positive trades	High
Average Return (%)	Average return of all trades	High
Average Positive Return (%)	Average return of positive trades	High
Average Negative Return (%)	Average return of negative trades	Low
Max Drawdown (%)	Maximum portfolio loss (peak to trough)	Low
Max Drawdown Duration (Days)	Duration of max drawdown	Low
Sharpe Ratio	Ratio of return average to stddev	High
Skew	Asymmetry of return distribution	High
Kurtosis	Tailedness of return distribution	Low

Table 4.2: Trading Performance Metrics

4.4 Fully Connected Neural Network Modelling

4.4.1 Data Pre-Processing

Before applying the TT models, it is important to have a baseline of reference for comparison purposes. Therefore, This section will first construct and train a simple fully-connected neural network.

As discussed in the previous section, the overall dataset is an order-4 tensor $\mathcal{D} \in \mathbb{R}^{3000\text{x}20\text{x}6\text{x}4}$ with the modes corresponding to time steps, new features, assets per asset class and asset classes, while the label is simply a vector $\mathbf{y} \in \mathbb{R}^{3000}$ which contains the next day returns for the JPY/USD currency pair. However, to train a standard fully-connected model, the data requires further processing.

Firstly, since the standard neural network works with matrices only, $\mathcal{D}$ is matricized along mode-1 (reshaped) to form $\mathbf{X} \in \mathbb{R}^{1000\text{x}480}$ such that mode-1 corresponds to time steps (number of samples) while mode-2 corresponds to various features of all assets. Then, $\mathbf{X}$ is split in $\mathbf{X}_{tr}$ and $\mathbf{X}_{te}$ using a training-to-testing ratio of 90%. It is important to highlight that the train-test split is done without shuffling since the samples (the time steps) are not necessarily i.i.d.

In addition, the training set is scaled using a standard scaler such that all features have mean of 0 and unit standard deviation to better fit the non-linear activation regions of the neural network. The same set of features of the test set are scaled using the scaling factors of the training set so that no information is leaked.

Therefore, the neural network is learning the function $f(\mathbf{x}_s) = \hat{\mathbf{y}} \in \{100, 010, 001\}$ where $\mathbf{x}_s \in \mathbb{R}^{1\text{x}480}$ is a sample from $\mathbf{X}$.

4.4.2 Model Architecture

To solve this classification problem, the following 2-layers model in the following table is constructed. This particular shallow architecture is used following the fact that with a sufficiently

high number of neurons (10^5 in this case), a two-layers neural network with sigmoid non-linearity can approximate virtually any decision boundary for classification [23].

It is important to note that very-wide shallow neural networks is not traditionally used due to high computational complexity, memory requirements and over-fitting risks [17]. However, since the TT-train format can potentially solve all of these issues, it is possible to compare the performance of this architecture with its TT counterpart later on.

Layer	Weight Matrix Shape	Bias Vector Shape	Parameters	Activation
FCL 1	$(480, 10^5)$	$(10^5, 1)$	48100000	Sigmoid
FCL 2	$(10^5, 3)$	$(3, 1)$	300003	Softmax

Table 4.3: Fully-Connected Model Architecture

4.4.3 Training Results and Analysis

The model is trained using stochastic gradient descent with a learning rate of 10^{-5} for 20 epochs with a batch size of 66 (equivalent to number of trading days in a quarter) with the loss function being the categorical cross-entropy. In terms of computational complexity, the training took 513.353 seconds in total.

The training/validation curves of the model is shown in figure 4.5, where the cross-entropy loss decreases monotonically for both training and validation data, suggesting that the model complexity is too low for modelling the given data since it doesn't over-fit. The under-fit of the model is also suggested by the weight matrix values which don't vary much over epochs as it is shown in the figure 4.6 (It usually follows an exponentially decreasing curve).

Finally, the backtest of the model is ran over the testing period, where the algorithm is assumed to update its position on the asset on a daily basis, and the size of its bet is proportional to the confidence of the model which varies from 0 to 1.

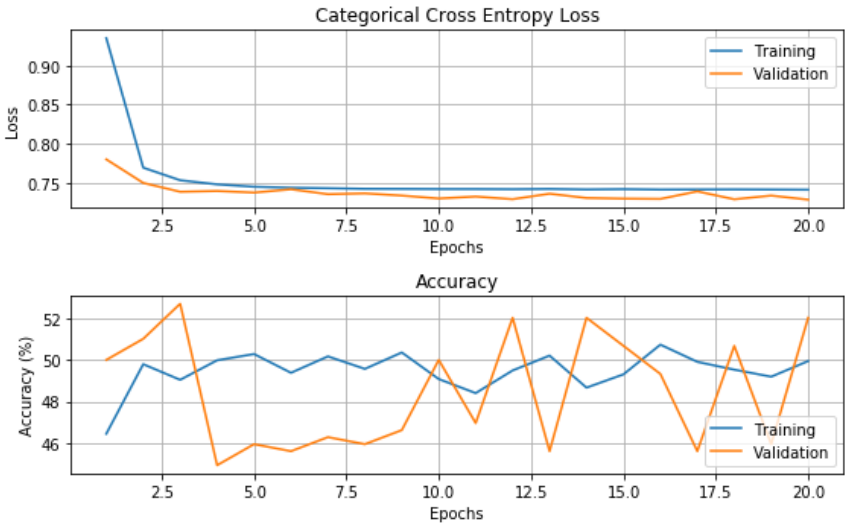


Figure 4.5: FC Model Training Curves

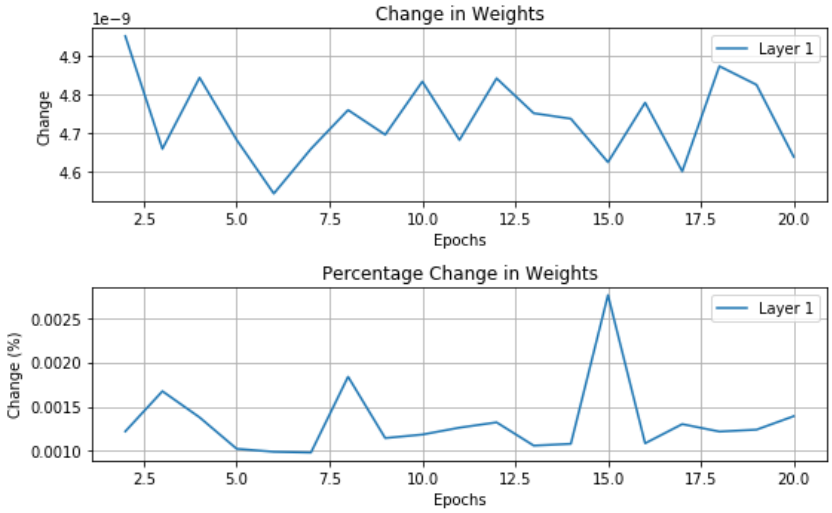


Figure 4.6: FC Model Weights Evolution

The tables below summarize the training results and the trading performance metrics of this model, while figure 4.4.3 shows the cumulative return of the backtest for the model and for the long-only strategy. Overall, the model performed poorly, achieving near 50% in terms of directional accuracy and a total return of 1.60% in trading, which is not surprising since the fully-connected model isn't suited for time series modelling.

Training Loss	Test Loss	Training Accuracy	Test Accuracy
0.7414	0.7287	49.94%	50.15%

Table 4.4: Fully-Connected Model Training Results

Metric	Value
Total Return (%)	1.602299
Winning Trades (%)	50.1520
Average Return (%)	0.004870
Average Positive Return (%)	0.150539
Average Negative Return (%)	-0.144327
Max Drawdown (%)	-3.058614
Max Drawdown Duration (Days)	134
Sharpe Ratio	0.025585
Skew	0.061947
Kurtosis	0.630156

Table 4.5: Fully-Connected Model Trading Performance

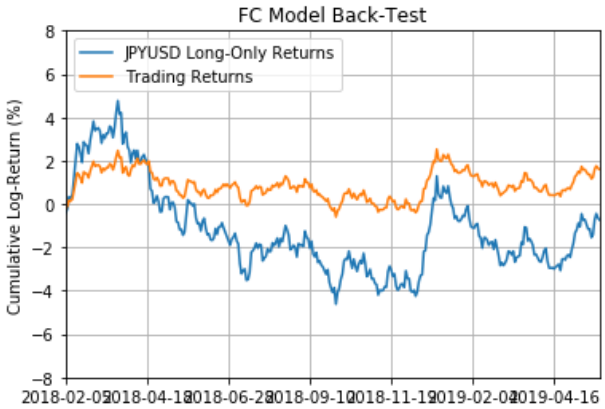


Figure 4.7: FC Model Backtest

4.5 TT Fully-Connected Neural Network Modelling

4.5.1 Data Pre-Processing

Unlike the standard neural network model, the TT model can process the dataset in its natural tensor form $\mathcal{D} \in \mathbb{R}^{3000\text{x}20\text{x}6\text{x}4}$ so that no matricization is needed. Other procedures such as the train-test split along the time-mode, data scaling and labelling are kept the same.

4.5.2 TT-Ranks and Model Performance

Since the TTD of the weight tensor is essentially a compression method, depending on the TT-ranks, several TT models exist that can approximate the original neural network architecture, each with different complexity and performance trade offs. This is shown in figure 4.8 where several values are computed against varying TT-ranks $(1, r, r, r, r, 1)$ for $r = 1, 2, ..., 10$, while the table below shows the trading performance of corresponding models.

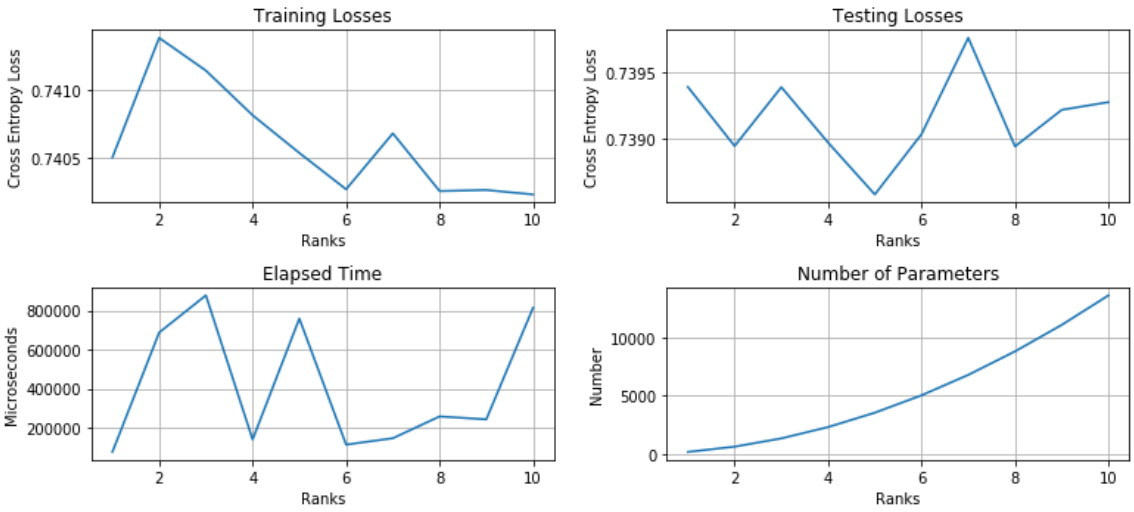


Figure 4.8: TT FC Model Complexity

The cross-entropy losses and the elapsed time to train the model don't vary much as a function of r. This is expected since the fully-connected model is not adequate to model the given data so that better approximation of the original weight matrix doesn't improve the performance. The

storage complexity instead is exponential in r, which is congruent with the super-compression characteristics of the TT format [3].

The under-fit of the model is also shown in the table below as the trading performance is negative for most r. Note that greater accuracy doesn't always imply better returns, since it is possible to be correct about small market fluctuations but wrong about the large ones.

	1	2	3	4	5	6	7	8	9	10
Total Return (%)	-0.998512	0.389235	0.353684	1.584443	0.245985	-1.137820	-0.356739	-2.973803	-0.361863	-3.531295
Winning Trades (%)	0.489362	0.501520	0.501520	0.510638	0.507599	0.489362	0.489362	0.468085	0.489362	0.483283
Average Return (%)	-0.003035	0.001183	0.001075	0.004816	0.000748	-0.003458	-0.001084	-0.009039	-0.001100	-0.010733
Average Positive Returns (%)	0.144363	0.145580	0.145355	0.144988	0.139259	0.140351	0.146725	0.140508	0.144383	0.133628
Average Negative Returns (%)	-0.146915	-0.146779	-0.146769	-0.144136	-0.144719	-0.143845	-0.145330	-0.143093	-0.143076	-0.148372
Max Drawdown (%)	-4.752956	-3.019823	-3.043754	-2.984434	-2.597457	-5.840160	-4.810632	-6.537370	-4.726949	-8.852064
Drawdown Duration (Days)	272.000000	62.000000	62.000000	62.000000	32.000000	183.000000	134.000000	272.000000	134.000000	149.000000
Sharpe Ratio	-0.016169	0.006273	0.005702	0.025823	0.004081	-0.018843	-0.005754	-0.049433	-0.005930	-0.058983
Skew	0.084046	-0.078230	-0.090314	-0.054830	0.025724	0.022090	0.085359	-0.097439	0.082939	-0.096525
Kurtosis	0.607700	0.620504	0.634872	0.631778	0.633802	0.664500	0.623357	0.612376	0.631203	0.618324

Figure 4.9: Trading Performance with Varying TT-Ranks (TT FC)

4.5.3 Model Architecture

For the best performing TT model by total return, the first layer of the fully-connected model is replaced by the TT layer. The original weight matrix $\mathbf{W} \in \mathbb{R}^{N \text{x} M}$ where $M = 480$ and $N = 10^5$ are factorized as M = 2x2x5x6x4 and N = 10x10x10x10x10. This factorization preserves physical significance where the first 3 cores correspond to features while the last 2 cores correspond to assets per asset class and asset classes. Finally, the TT-rank for this model is $(1, 4, 4, 4, 4, 1)$, this layer stores $\sum_{k=1}^{d} m_k n_k r_{k-1} r_k = 2320$ parameters (for $d = 5$ cores) instead of $\prod_{k=1}^{d} m_k n_k = MN = 480\text{x}10^5$, which is a reduction by a factor of $21\text{x}10^3$.

Layer	Weight Matrix	Bias	Parameters	Activation
Fully-Connected 1	N.A.	$(10^5, 1)$	102320	Sigmoid
Fully-Connected 2	$(10^5, 3)$	$(3, 1)$	300003	Softmax

Table 4.6: TT Fully-Connected Model Architecture

4.5.4 Training Results and Analysis

The model is trained using the same hyper-parameters to make the comparison sensible. Overall, the training took 169.285 seconds in total, which is about 3 times faster.

The training curves of this model are similar to the standard architecture and shows underfitting characteristics (figure 4.10). In terms of weights evolution, it is possible to partition the weights into individual cores and analyze their evolution over epochs (figure 4.11) which retains physical significance.

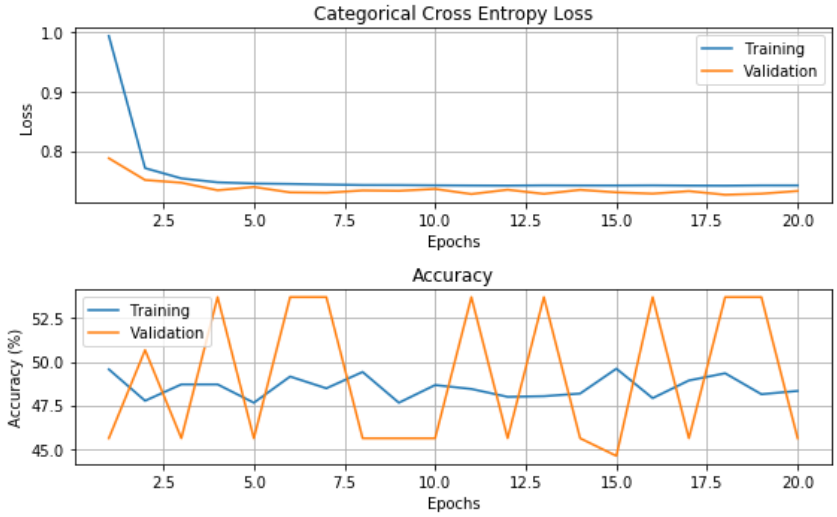


Figure 4.10: TT FC Model Training Curves

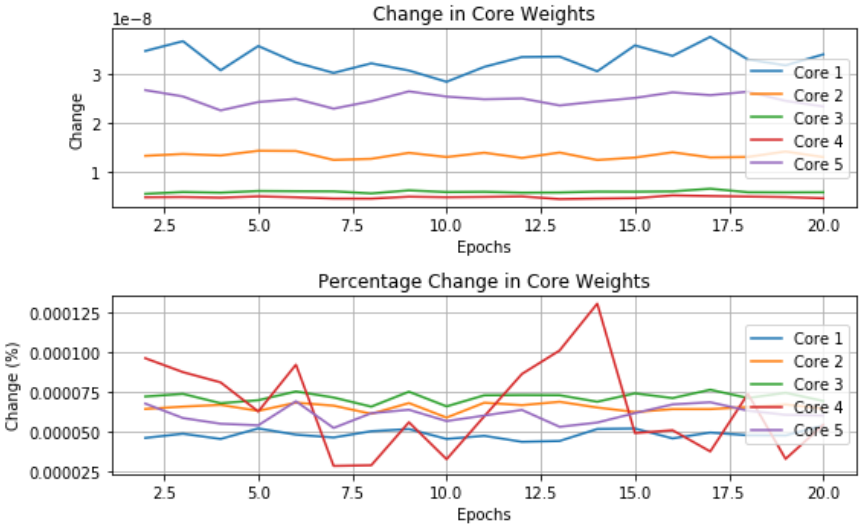


Figure 4.11: TT FC Model Weights Evolution

Overall, the cores that are updated the most via back-propagation can be interpreted to contain most important information. In this case, the cores 1,5 and 2 are the most changed ones, which relate to the returns, asset classes and pricing information respectively.

The backtest of this model performed similarly to the standard fully-connected model despite the reduction in weight matrix by a factor of $21\text{x}10^3$. The training results and trading metrics are summarized in the table below, while the trading performance is plotted in figure 4.12.

Training Loss	Test Loss	Training Accuracy	Test Accuracy
0.7421	0.7389	49.57%	51.06%

Table 4.7: TT Fully-Connected Model Results

Metric	Value
Total Return (%)	1.584443
Winning Trades (%)	51.0638
Average Return (%)	0.004816
Average Positive Return (%)	0.144988
Average Negative Return (%)	-0.144136
Max Drawdown (%)	-2.984434
Max Drawdown Duration (Days)	62
Sharpe Ratio	0.025823
Skew	-0.054830
Kurtosis	0.631778

Table 4.8: TT Fully-Connected Model Trading Performance

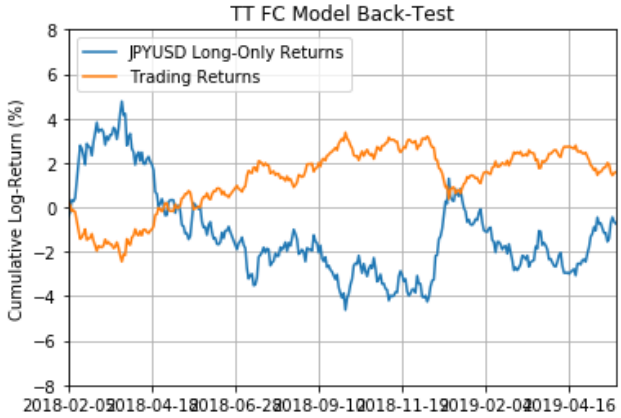


Figure 4.12: TT FC Model Backtest

4.6 Recurrent Neural Network Modelling

4.6.1 Data Pre-Processing

The standard fully-connected model lacks expressivity and is not complex enough to capture the temporal dependencies that exist in the dataset. This section will instead apply a recurrent neural network to model the time-series.

The recurrent neural network requires the input data to be shaped differently. For instance, each sample in the dataset should be a matrix with mode-1 corresponding to time steps and mode-2 corresponding to features. For this particular model, the time step is set to be 10 (trading days in 2 weeks) and is sampled from the same dataset from section 4.4 on a rolling basis (figure 4.13).

Therefore, the function that the recurrent model is learning is $f(\mathbf{X}_s) = \hat{\mathbf{y}} \in \{100, 010, 001\}$ where $\mathbf{X}_s \in \mathbb{R}^{10\text{x}480}$ is a sample from the dataset $\mathcal{X} \in \mathbb{R}^{2990\text{x}10\text{x}480}$. Finally, other pre-processing procedures such as the train-test split along the time-mode, data scaling and labelling are all kept the same.

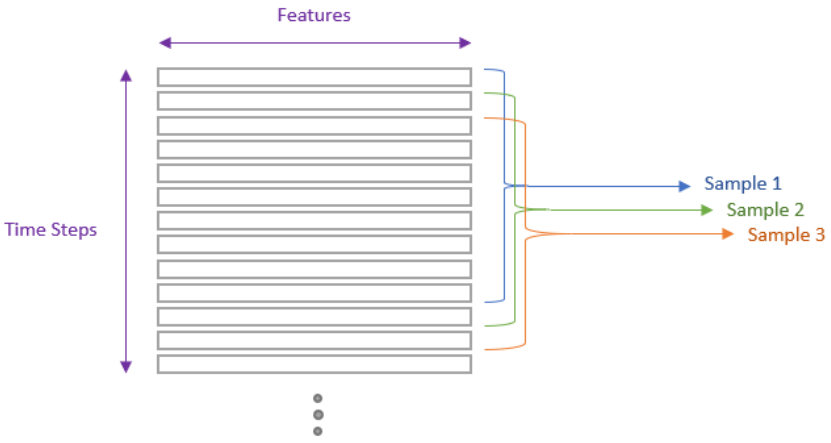


Figure 4.13: Recurrent Model Data Sampling

4.6.2 Model Architecture

Following a similar approach, a 2-layers recurrent model described in the following table is constructed. The number of units in the recurrent layer is limited to only $1024 = 4^5$, but due to higher expressivity of the recurrent neural network [36], the model exhibits over-fitting characteristics even for a shallow architecture with fewer units.

Layer	Weight Matrix	Recurrent Matrix	Bias	Parameters	Activation
Recurrent	$(480, 1024)$	$(1024, 1024)$	$(1024, 1)$	1541120	Tanh
Fully-Connected	$(1024, 3)$	N.A.	$(3, 1)$	3075	Softmax

Table 4.9: Recurrent Model Architecture

4.6.3 Training Results and Analysis

The model is trained using the same hyperparameters as in section 4.4, with the training taking 169.285 seconds in total. Figure 4.14 shows the training curves of the model where the cross-entropy losses diverge for the training and validation set. This indicates that unlike the fully-connected model, the model is expressive and complex enough to capture the temporal relationship in the data but is over-fitting. This is also shown by the weight matrix which follows a classical decreasing curve in both the feature mapping weight matrix $\mathbf{W}^{xh}$ and recurrent mapping weight matrix $\mathbf{W}^{hh}$.

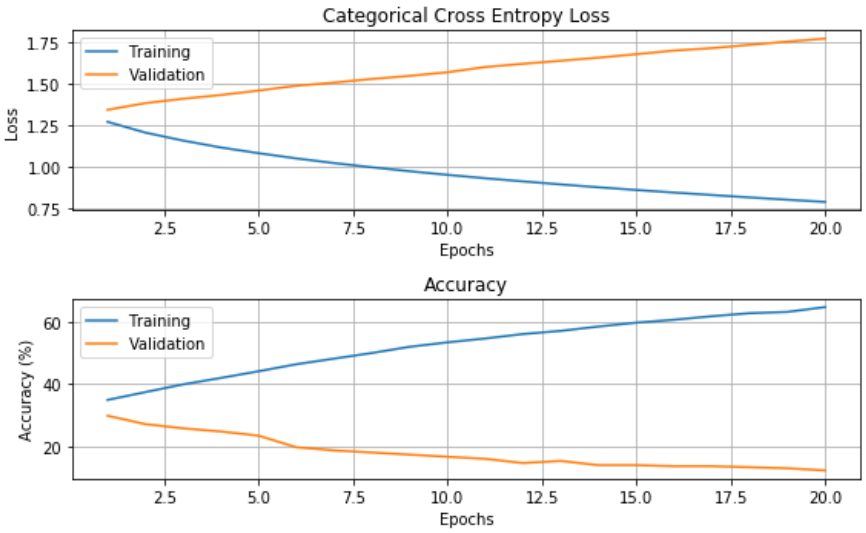


Figure 4.14: Recurrent Model Training Curves

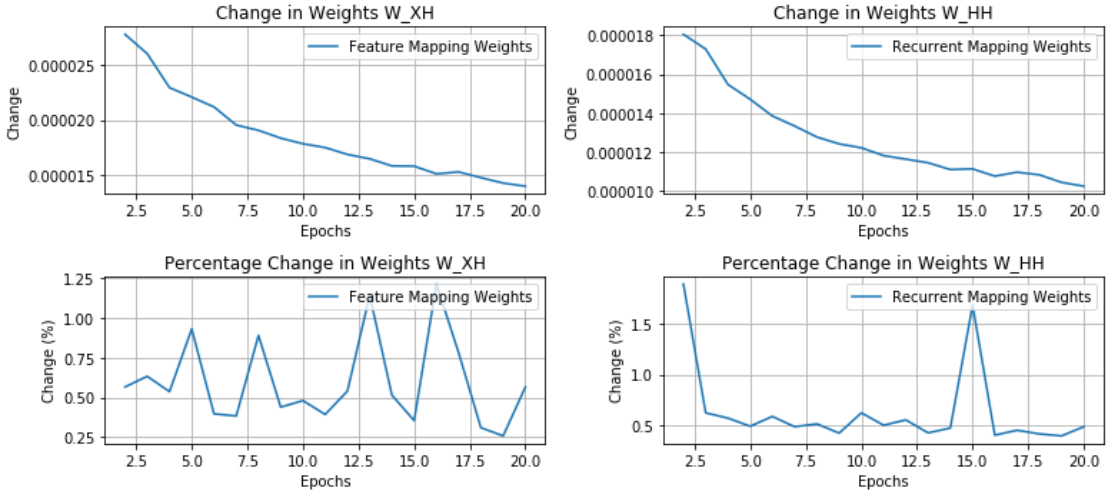


Figure 4.15: Recurrent Model Weights Evolution

Overall, the recurrent model's forecasting results are worse than the fully-connected model, both in terms of accuracy (21.32%) and total return (-3%), which is not surprising due to the model over-fitting. The backtest of this model is shown in figure 4.16, while the training and trading performance metrics are summarized in the table below.

Training Loss	Test Loss	Training Accuracy	Test Accuracy
0.7923	1.7690	64.83%	21.32%

Table 4.10: Recurrent Model Results

Metric	Value
Total Return (%)	-2.995116
Winning Trades (%)	21.3166
Average Return (%)	-0.009389
Average Positive Return (%)	0.164507
Average Negative Return (%)	-0.196967
Max Drawdown (%)	-4.150816
Max Drawdown Duration (Days)	231
Sharpe Ratio	-0.058657
Skew	-0.305198
Kurtosis	4.981242

Table 4.11: Recurrent Model Trading Performance

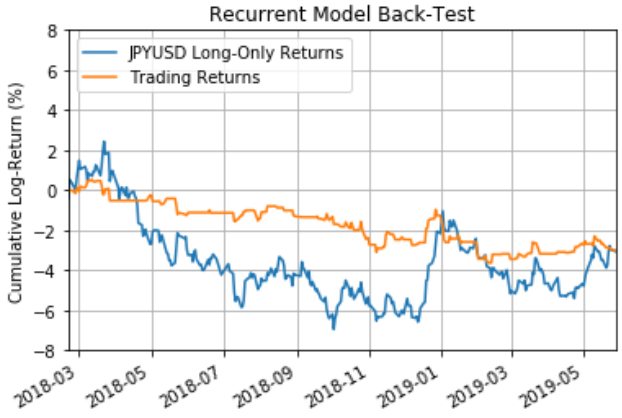


Figure 4.16: Recurrent Model Backtest

4.7 TT Recurrent Neural Network Modelling

4.7.1 Data Pre-Processing

This model is trained using stochastic gradient descent with a learning rate of 10^{-3}, while all other pre-processing procedures such as the train-test split along the time-mode, data scaling and labelling are kept the same.

4.7.2 TT-Ranks and Model Performance

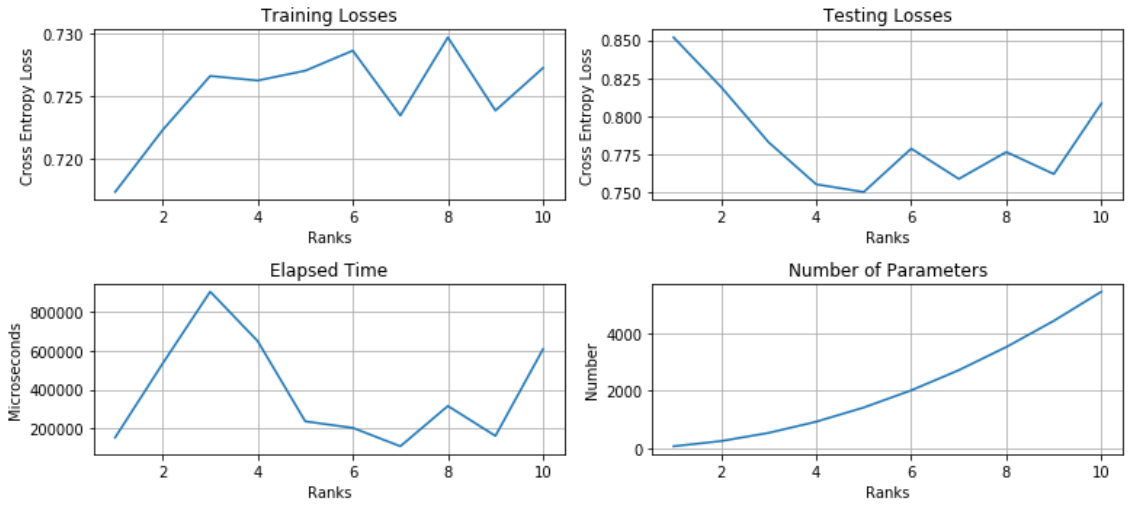


Figure 4.17: TT Recurrent Model Complexity

The performance-complexity trade-off of the TT recurrent model is shown in figure 4.17 where several values are computed against varying TT-ranks $(1, r, r, r, r, 1)$ for $r = 1, 2, ..., 10$.

It is interesting to note that the testing cross-entropy loss form a U-shape as r increases. This is not surprising since larger r implies less compression which means that the regularization power decreases. With less regularization, there are more chances of over-fitting, hence resulting in worse performance at higher r. The varying regularization power as function of TT-ranks is also shown in the table below, where the trading performance tends to be worse for models that have too large or too small values of r in terms of accuracy, while the accuracy tends to be above 52% for r between 4 and 8.

	1	2	3	4	5	6	7	8	9	10
Total Return (%)	2.706154	0.371709	2.398863	0.927408	-0.161603	6.726204	3.053046	2.758548	1.383401	2.246998
Winning Trades (%)	0.495298	0.482759	0.495298	0.523511	0.501567	0.529781	0.536050	0.520376	0.504702	0.501567
Average Return (%)	0.008483	0.001165	0.007520	0.002907	-0.000507	0.021085	0.009571	0.008647	0.004337	0.007044
Average Positive Returns (%)	0.190285	0.158394	0.172720	0.150579	0.150442	0.162846	0.156231	0.165860	0.162096	0.158920
Average Negative Returns (%)	-0.173158	-0.149198	-0.157537	-0.162546	-0.155335	-0.141461	-0.163190	-0.165162	-0.159446	-0.148591
Max Drawdown (%)	-3.794850	-2.075688	-3.657522	-3.439662	-4.028415	-1.756851	-2.395287	-3.126632	-4.158285	-2.278618
Drawdown Duration (Days)	98.000000	87.000000	82.000000	138.000000	104.000000	28.000000	51.000000	36.000000	88.000000	66.000000
Sharpe Ratio	0.036117	0.005792	0.035533	0.014265	-0.002553	0.105584	0.046730	0.039837	0.020828	0.035072
Skew	-0.024089	0.210376	-0.148928	0.160391	0.133135	-0.114495	0.082072	-0.189876	0.250018	0.094612
Kurtosis	0.515727	1.050198	0.587937	1.150511	0.944104	1.113849	0.679665	1.218101	0.885614	1.174760

Figure 4.18: Trading Performance with Varying TT-Ranks (TT RNN)

4.7.3 Model Architecture

For the best performing TT recurrent model by total return, the features mapping weight matrix of the recurrent layer $\mathbf{W}^{xh}$ is replaced by the TT format. The original matrix $\mathbf{W}^{xh} \in \mathbb{R}^{N \text{x} M}$ where $M = 480$ and $N = 4^5$ is factorized such that $M = 2\text{x}2\text{x}5\text{x}6\text{x}4$ and $N = 4\text{x}4\text{x}4\text{x}4\text{x}4$. The TT-ranks for this particular model are $(1, 6, 6, 6, 6, 1)$, which means that $\mathbf{W}^{xh}$ is replaced by $\sum_{k=1}^{d} m_k n_k r_{k-1} r_k = 2016$ parameters (for $d = 5$ cores) instead of $\prod_{k=1}^{d} m_k n_k = MN = 491520$, which is a reduction by a factor of 244.

Layer	Weight Matrix	Recurrent Matrix	Bias	Parameters	Activation
Recurrent	N.A.	$(1024, 1024)$	$(1024, 1)$	1051616	Tanh
Fully-Connected	$(1024, 3)$	N.A.	$(3, 1)$	3075	Softmax

Table 4.12: TT Recurrent Model Architecture

4.7.4 Training Results and Analysis

The model is trained using the same hyper-parameters to make the comparison sensible and took 134.189 seconds in total, which is faster than the standard recurrent model.

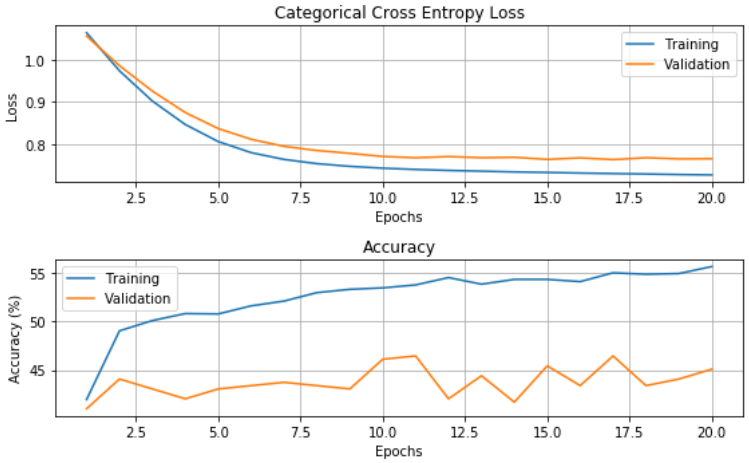


Figure 4.19: TT Recurrent Model Training Curves

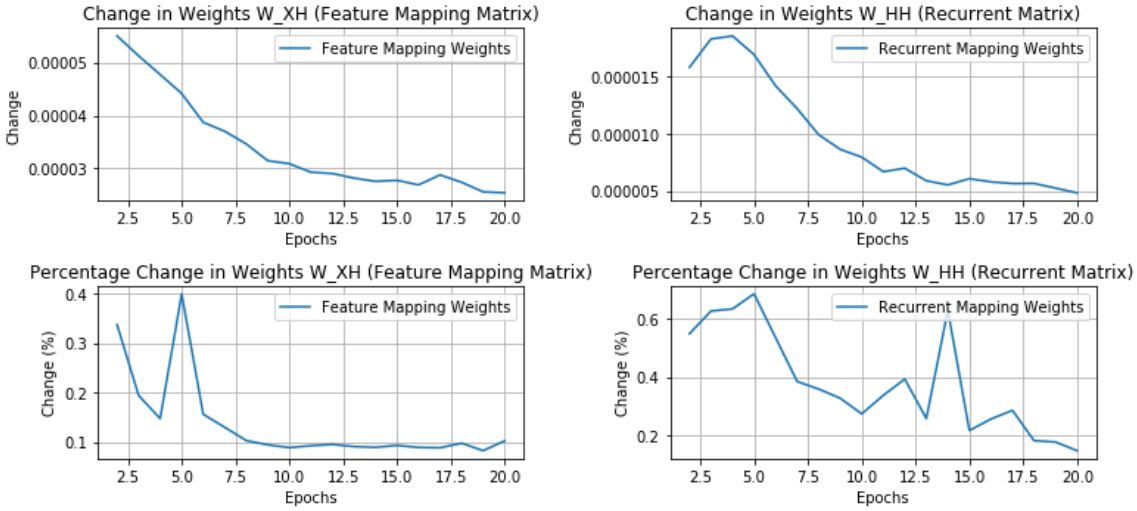


Figure 4.20: TT Recurrent Model Weights Evolution

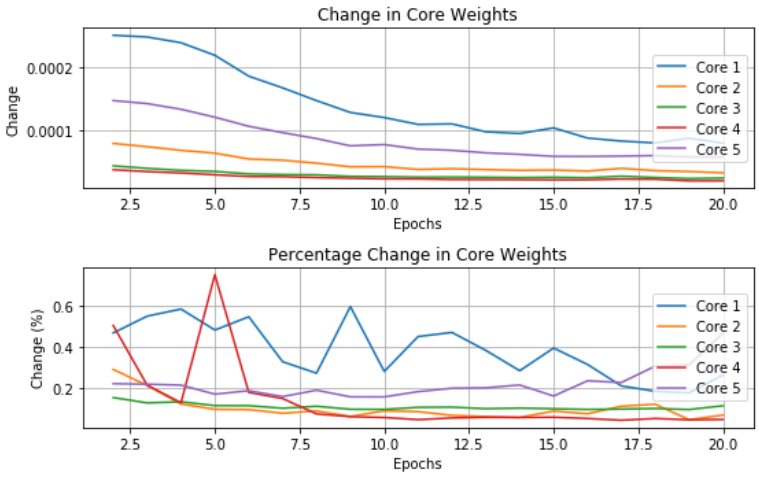


Figure 4.21: TT Recurrent Model Cores Weights Evolution

The training/validation curves (figure 4.19) of the TT-RNN model show no signs of over-fitting due to the regularization of the TT format [6]. For instance, the loss is monotonically decreasing for both the training and the validation set, while the accuracy improves over epochs for both too. Therefore, the TT Model not only compresses the model but it also improves the performance and robustness due to the inherent regularization effects.

The regularized and compressed model exhibits smooth changes in its weights as well, suggesting that the regularization did not compromise the complexity and expressivity of the model. This is shown in figure 4.20.

In addition, it is possible to break-down the weights into individual cores and analyze their evolution over epochs, which is shown in figure 4.21. Similar to the TT fully-connected model, the weights of cores 1, 5, 2 are the ones that are updated the most, suggesting that these 3 cores and their corresponding features convey most information regarding the feature mapping.

The final training and trading performance results of the TT model is summarized in the table below, while figure 4.22 compares the trading cumulative returns against a long-only strategy. The model demonstrates an improvement in performance not only compared to the baseline model but also to the standard recurrent model, achieving 6.7% in total return and near 53% in directional accuracy despite a significant reduction in parameters.

It is interesting to note that despite the good trading performance compared to other models, the test cross-entropy value is not as ideal. This is fundamentally due to the set up of the classification problem and the choice of loss function, which will be discussed in more details in the conclusion section.

Training Loss	Test Loss	Training Accuracy	Test Accuracy
0.7264	0.7765	56.53%	52.98%

Table 4.13: TT Recurrent Model Results

Metric	Value
Total Return (%)	6.726204
Winning Trades (%)	52.9781
Average Return (%)	0.021085
Average Positive Return (%)	0.162846
Average Negative Return (%)	-0.141461
Max Drawdown (%)	-1.756851
Max Drawdown Duration (Days)	28
Sharpe Ratio	0.105584
Skew	-0.114495
Kurtosis	1.113849

Table 4.14: TT Recurrent Model Trading Performance

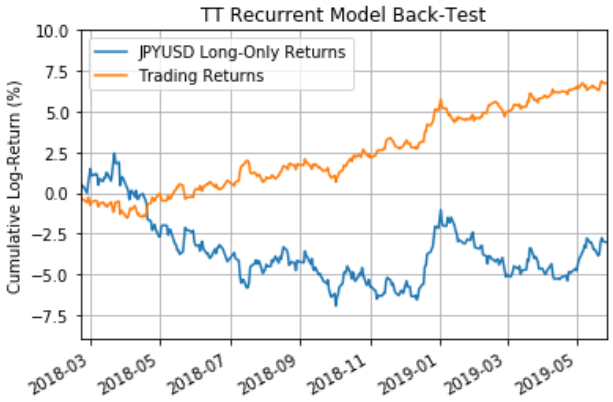


Figure 4.22: TT Recurrent Model Backtest

4.8 Evaluation of Results

The Tensor-Train format can reduce the memory requirements of various models by a significant degree. In the case where the neural network to be approximated under-fits the data, the TT format compresses the storage requirement without significant drop in performance. In the case where the model over-fits data, the TT format offers regularization characteristics in addition to the compression advantages.

When the fully-connected models and the recurrent models are tensorized using the TT format, the above mentioned characteristics are demonstrated to be true. For the modelling of financial time series, the TT recurrent model combined the expressive power of recurrent neural networks together with the regularization effects of the tensor-train format, achieving the best results among all architectures. This is shown in the tables below which summarize the modelling and trading performance of all above mentioned models.

Metric	FC	TT FC	Recurrent	TT Recurrent
Training Cross-Entropy Loss	0.7414	0.7421	0.7923	0.7264
Testing Cross-Entropy Loss	0.7287	0.7389	1.7690	0.7765
Training Accuracy (%)	49.94	49.57	64.83	56.53
Testing Accuracy (%)	50.15	51.06	21.32	52.98

Table 4.15: Training Results Summary

Metric	FC	TT FC	Recurrent	TT Recurrent
Total Return (%)	1.602299	1.584443	-2.995116	6.726204
Winning Trades (%)	50.1520	51.0638	21.3166	52.9781
Average Return (%)	0.004870	0.004816	-0.009389	0.021085
Average Positive Return (%)	0.150539	0.144988	0.164507	0.162846
Average Negative Return (%)	-0.144327	-0.144136	-0.196967	-0.141461
Max Drawdown (%)	-3.058614	-2.984434	-4.150816	-1.756851
Max Drawdown Duration (Days)	134	62	231	28
Sharpe Ratio	0.025585	0.025823	-0.058657	0.105584
Skew	0.061947	-0.054830	-0.305198	-0.114495
Kurtosis	0.630156	0.631778	4.981242	1.113849

Table 4.16: Trading Performance Summary

Chapter 5

Conclusion

5.1 Summary of Project Achievements

The tensor-train recurrent neural network demonstrated strong compression and regularization capabilities. In addition to striking a balance between modelling power and complexity, the TT format offers an additional layer of interpretability by drawing links across the data modality and structure, which helps overcoming the black box nature of neural networks.

In the case of tensorizing fully connected models, model parameters are reduced by a factor of $21\text{x}10^3$ with a change in classification accuracy of only 0.09 %. The tensor-train recurrent model instead demonstrated superior generalization abilities than the standard architecture, achieving the best out of sample classification rate of 52.98% and trading returns of 6.73%. However, the regularization abilities of the TT-format must be balanced with its compression level since excessive compression diminishes the model expressivity which can negatively impact the performance. For this reason, the compression factor for the recurrent model is only 244.

Finally, by analyzing the evolution of TT-cores, it is possible to establish the relative importance of various input data modes. For the financial application carried out in this project, the information across the market data and asset classes offered most information in terms of modelling.

5.2 Future Work

Overall, the project has met its aims, but there are many aspects that can be further explored. For instance, in terms of compression, it is possible to further reduce the computational complexity of the model by decomposing the input in TT format [17]. The implementation of the TT-matrix by TT-vector multiplication would require a different algorithm and back-propagation procedure [4], which can be explored as future work.

In addition, a non-formal interpretation of TT-cores in neural networks is presented in the report, which provides an intuition for the links that exist between the data structure and the way it is processed. However, a formal mathematical proof should be investigated which can lead to new insights in terms of interpretability. Finally, it is possible to explore other well understood decomposition methods such as CPD to see if the interpretation remains consistent.

Furthermore, other qualities of a given network such as expressive efficiency and inductive bias can also be analysed quantitatively through tensor decomposition to design better networks. For instance, some work has shown that the performance of the convolutional neural networks is more dependent on the architectural characteristics rather than the choices of activation functions [37] [38]. Representation power in terms of architecture can be quantitatively analyzed through an equivalent hierarchical tensor decomposition of the network such as TTD and can be applied to recurrent neural networks.

Finally, the final model is a simplistic model which is not adequate for real financial application. For instance, although the final model exhibits consistent classification rates for certain range of TT-ranks (see figure 4.18), the trading performance is not as robust and varies greatly. This is due to the definition of the loss function which does not optimize the mean-variance characteristics of returns of an ideal portfolio, but rather minimizes the cross-entropy value. The focus on minimizing classification loss doesn't account for the tails of the return distribution which can ruin a portfolio and hence produces non consistent trading performances. In addition to a better loss function, further learning on market regime detection and portfolio construction for risk management is needed before the model is practically viable.

Bibliography

[1] A. Cichocki. "Era of big data processing: A new approach via tensor networks and tensor decompositions". In *Proceedings of 2013 Int. Workshop on Smart Info-Media Systems in Asia, SISA-2013*, Nagoya, Japan, Oct. 1, 2013.

[2] R. Caruana, S. Lawrence, and C. L. Giles. "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping". In *Advances in neural information processing systems*, pages 402–408, 2001.

[3] B. N. Khoromskij. "O(dlog n)-quantics approximation of n-d tensors in high-dimensional numerical modeling". *Constructive Approximation*, 34(2):257–280, 2011.

[4] I. V. Oseledets. "Tensor-train decomposition". *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[5] Y. Yang, D. Krompass, and V. Tresp. "Tensor-train recurrent neural networks for video classification". In *ICML, 38913900*, 2017.

[6] X. Cao and G. Rabusseau. "Tensor regression networks with various low-rank tensor approximations". *CoRR abs/1712.09520*, 2017.

[7] F. Black and M. Scholes. "The pricing of options and corporate liabilities". *Journal of political economy*, 81(3):637–654, 1973.

[8] K. J. Kim. "Financial time series forecasting using support vector machines". *Neurocomputing*, 55(1-2):307–319, 2003.

[9] E. M. Azoff. *"Neural network time series forecasting of financial markets"*. John Wiley & Sons, Inc., 1994.

[10] F. Z. Xing, E. Cambria, and R. E. Welsch. "Natural language based financial forecasting: a survey". *Artificial Intelligence Review*, 50(1):49–73, 2018.

[11] J. Ye, R. Janardan, and Q. Li. "GPCA: an efficient dimension reduction scheme for image compression and retrieval". In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 354–363. ACM, 2004.

[12] E. Bingham and H. Mannila. "Random projection in dimensionality reduction: applications to image and text data". In *Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 245–250. ACM, 2001.

[13] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. "Beyond short snippets: Deep networks for video classification". In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015.

[14] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. "Long-term recurrent convolutional networks for visual recognition and description". In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015.

[15] 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". In *Proceedings of 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6655–6659. IEEE, 2013.

[16] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. "Speeding-up convolutional neural networks using fine-tuned CP-decomposition". *Computer Science*, 2014.

[17] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov. "Tensorizing neural networks". In *Proceedings of Advances in Neural Information Processing Systems*, pages 442–450, 2015.

[18] F. Verstraete, V. Murg, and J.I. Cirac. "Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems". *Advances in Physics*, 57(2):143–224, 2008.

[19] Y. Levine, O. Sharir, N. Cohen, and A. Shashua. "Bridging many-body quantum physics and deep learning via tensor networks". *arXiv preprint arXiv:1803.09780*, 2018.

[20] Y. Levine, D. Yakira, N. Cohen, and A. Shashua. "Deep learning and quantum entanglement: Fundamental connections with implications to network design". *arXiv preprint arXiv:1704.01552*, 2017.

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

[22] I. Kisil. "HOTTBOX, HOTTBOX-tutorials". https://github.com/hottbox/hottbox-tutorials.

[23] G. Cybenko. "Approximation by superpositions of a sigmoidal function". *Mathematics of Control, Signals, and Systems (MCSS)*, 5(4):455–455, 1992.

[24] H. Robbins and S. Monro. "A stochastic approximation method". *The annals of mathematical statistics*, pages 400–407, 1951.

[25] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski. "A theoretical framework for back-propagation". In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

[26] G. H. Skidmore. "Alternative asset classes: An introduction". *Belray Asset Management*, 2010.

[27] H. M. K. Mok. "Causality of interest rate, exchange rate and stock prices at stock market open and close in hong kong". *Asia Pacific Journal of Management*, 10(2):123–143, 1993.

[28] A. Ranaldo and P. Söderlind. "Safe haven currencies". *Review of finance*, 14(3):385–407, 2010.

[29] F. Chollet. "Keras". https://github.com/keras-team/keras.

[30] H. Wickham et al. "Reshaping data with the reshape package". *Journal of statistical software*, 21(12):1–20, 2007.

[31] R. A. Athale and W. C. Collins. "Optical matrix–matrix multiplier based on outer product decomposition". *Applied optics*, 21(12):2089–2090, 1982.

[32] Y. Fan, J. Jiao, Q. Liang, Z. Han, Y. Wei, et al. "The impact of rising international crude oil price on china's economy: an empirical analysis with cge model". *International Journal of Global Energy Issues*, 27(4):404, 2007.

[33] A. Estrella and M. Trubin. "The yield curve as a leading indicator: Some practical issues". *Current Issues in Economics and Finance*, 2006.

[34] Bloomberg L.P. Bloomberg terminal data, 2019.

[35] M. López de Prado. "The 10 reasons most machine learning funds fail". *Journal of Portfolio Management, Forthcoming*, 2018.

[36] V. Khrulkov, A. Novikov, and I. Oseledets. "Expressive power of recurrent neural networks". In *International Conference on Learning Representations*, 2017.

[37] N. Cohen, O. Sharir, Y. Levine, R. Tamari, D. Yakira, and A. Shashua. "Analysis and design of convolutional networks via hierarchical tensor decompositions". *CoRR, vol. abs/1705.02302*, 2017.

[38] N. Cohen, O. Sharir, and A. Shashua. "On the expressive power of deep learning: A tensor analysis". In *Proceedings of Conference on Learning Theory*, pages 698–728, 2016.

Appendix A

Safety, Legal and Ethical Issues

A.1 Safety Issues

Multiple safety issues are considered for this project which are discussed below.

- Data infrastructure safety

  The data used for the project will be provided by reliable sources (e.g. Bloomberg for financial data) and used with care such that they won't compromise any IT systems via the spreading of viruses or improper use of networks.

- Electrical safety

  There is no equipment involved in the project that constitutes an electrical hazard.

- Physical safety

  There is no large or fast-moving objects involved in the project that can cause harm to myself or others.

- Chemical safety

  There is no poisonous, irritant or allergenic material involved in the project.

- Fire safety

  There is no material or equipment involved in the project that constitutes a fire hazard.

- Biological safety

  There is no material involved in the project that can cause possible biological hazards.

- Animal safety

  There are no animals involved in the project.

- Study participant safety

  There are no study participants involved in the project.

A.2 Legal Issues

The only legal issue that can potentially arise from this project relates to the use of data. Given that the financial data will be provided by the Bloomberg terminal, the use of data will be subject to its term and services. Therefore, the use of data will be restricted to research purposes only and not for any commercial activity and will not be distributed to other parties.

A.3 Ethical Issues

There are no major ethical or moral issues related to the project.

Appendix B

Raw Data Plots

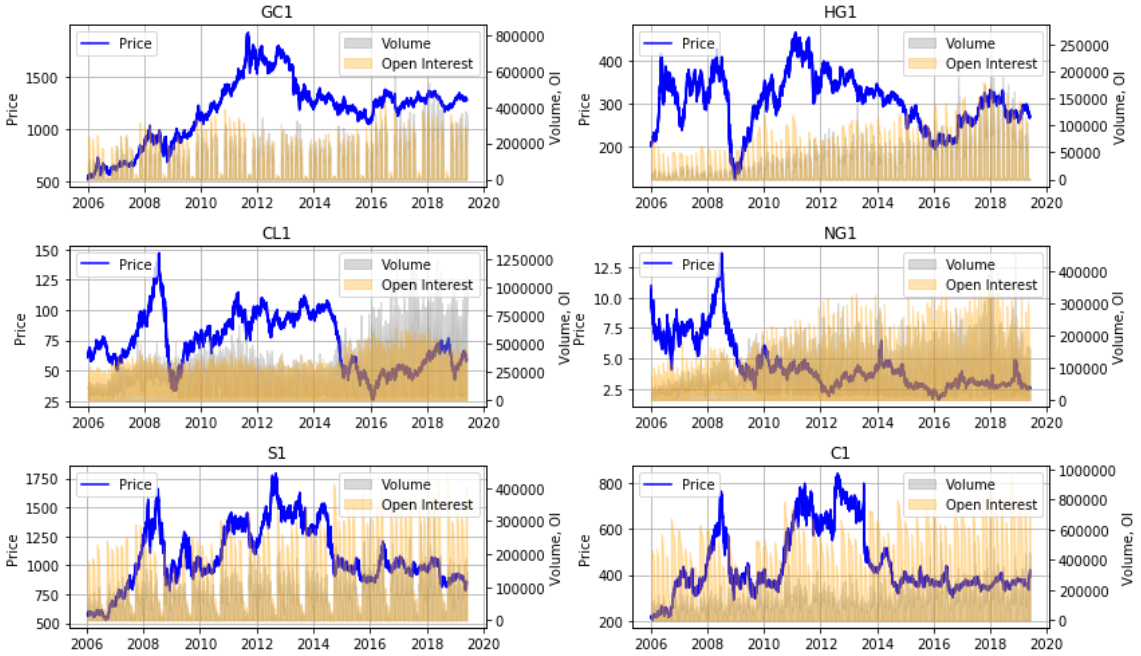


Figure B.1: Commodity Futures Graphs

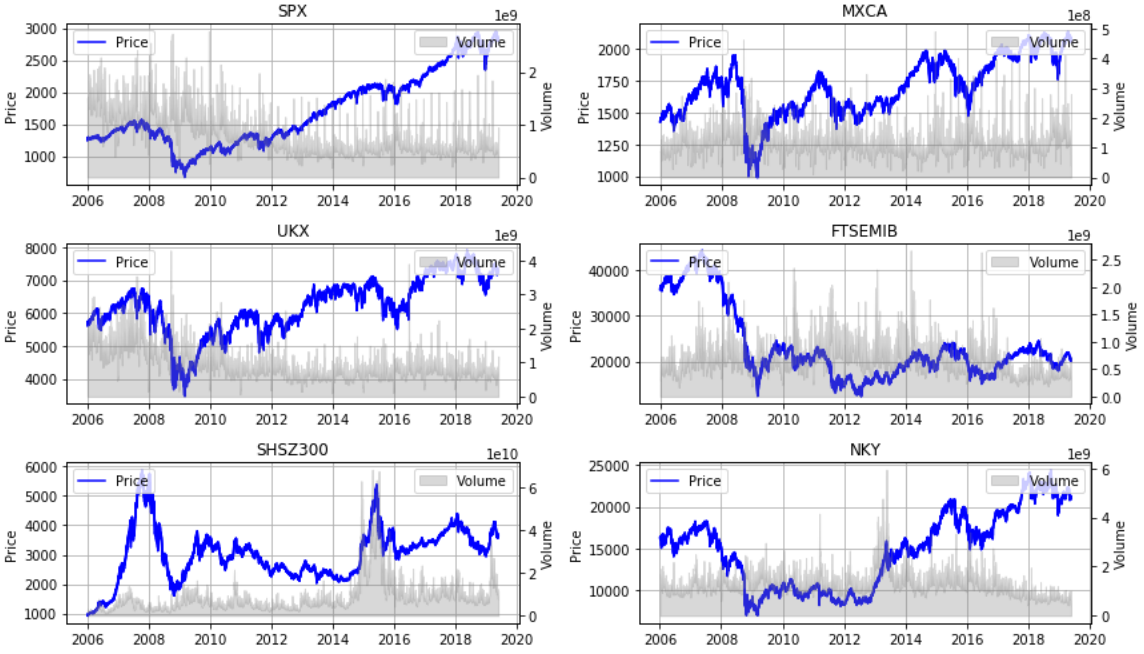


Figure B.2: Stock Market Indices Graphs

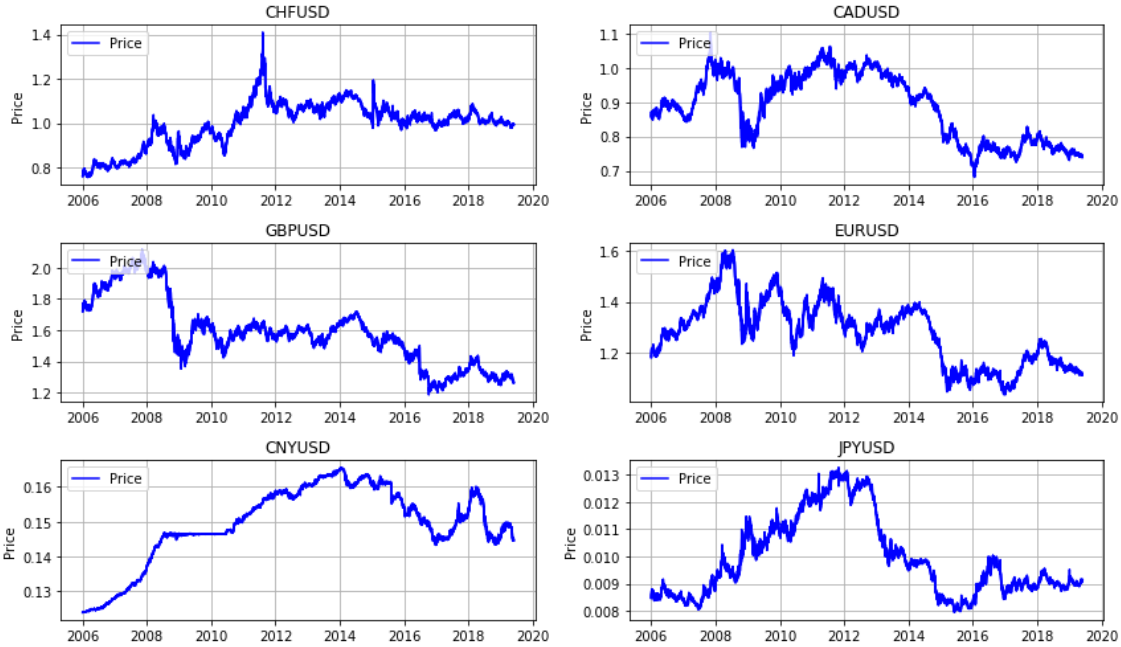


Figure B.3: FX Pairs Graphs

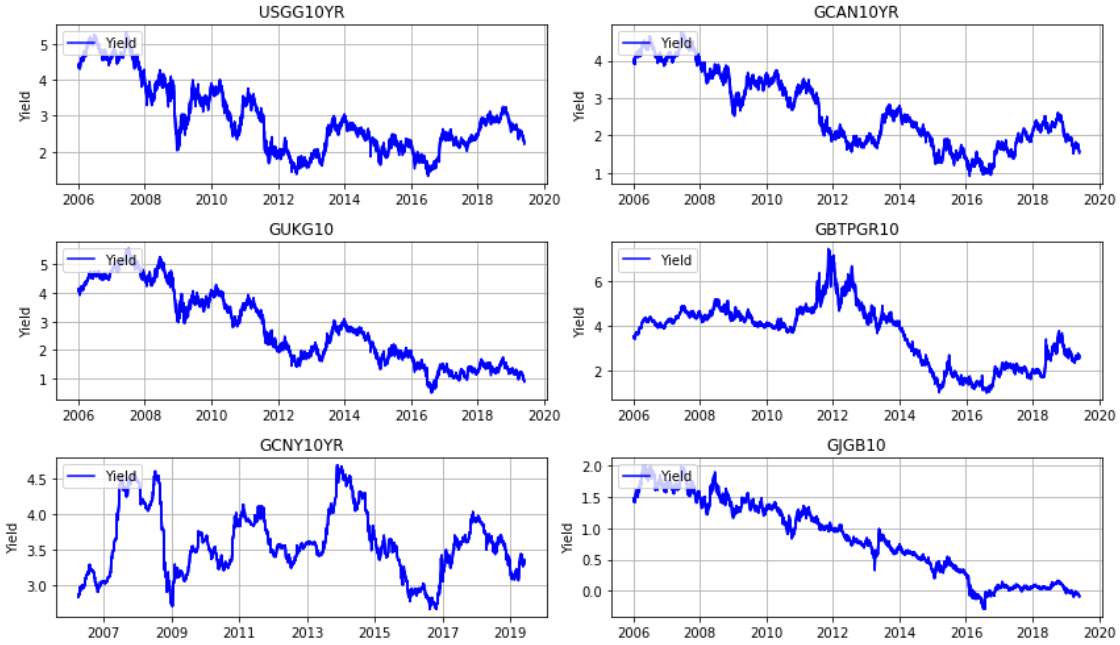


Figure B.4: 10-Year Yields Graphs

Appendix C

Code

The Jupyter notebooks for this project can be found on my github `https://github.com/gylx`