Tensor-Train Networks for Learning Predictive Modeling of Multidimensional Data

Michele Nazareth da Costa\textsuperscript{a}, Romis Attux\textsuperscript{a}, Andrzej Cichocki\textsuperscript{b,c}, João M. T. Romano\textsuperscript{a}

\textsuperscript{a}DSPCom laboratory, University of Campinas, Campinas-SP, Brazil
\textsuperscript{b}Skolkovo Institute of Science and Technology (SKOLTECH), CDISE, Moscow, Russian Federation
\textsuperscript{c}Nicolaus Copernicus University, 87-100 Torun, Poland

Abstract

Deep neural networks have attracted the attention of the machine learning community because of their appealing data-driven framework and of their performance in several pattern recognition tasks. On the other hand, there are many open theoretical problems regarding the internal operation of the network, the necessity of certain layers, hyperparameter selection etc. A promising strategy is based on tensor networks, which have been very successful in physical and chemical applications. In general, higher-order tensors are decomposed into sparsely interconnected lower-order tensors. This is a numerically reliable way to avoid the curse of dimensionality and to provide highly compressed representation of a data tensor, besides the good numerical properties that allow to control the desired accuracy of approximation. In order to compare tensor and neural networks, we first consider the identification of the classical Multilayer Perceptron using Tensor-Train. A comparative analysis is also carried out in the context of prediction of the Mackey-Glass noisy chaotic time series and NASDAQ index. We have shown that the weights of a multidimensional regression model can be learned by means of tensor networks with the aim of performing a powerful compact representation retaining the accuracy of neural networks. Furthermore, an algorithm based on alternating least squares has been proposed for approximating the weights in TT-format with a reduction of computational calculus. By means of a direct expression, we have approximated the core estimation as the conventional solution for a general regression model, which allows to extend the applicability of tensor structures to different algorithms.

Keywords: Tensor-Train network; multilinear regression model; multilayer perceptron network; neural network; time-series forecasting; supervised learning.

1. Introduction

Mathematical models, frequently based on assumptions about the nature of physical phenomenon, can be employed to simulate the relations between input and output parameters in order to interpret the inherent interactions and to predict a certain time-varying process. Hence, several modeling techniques have been developed with the aim of providing a more adequate understanding of complex systems. Time series analysis and forecasting are widely present in different fields, like econometrics, dynamic systems theory, statistics, pattern recognition, modeling physiological systems, climatology, biological system identification, among others Gooijer and Hyndman (2006); Box et al. (2015). The behavior of these phenomena is influenced by several factors that can affect the development of the system over time and, its interpretability through a proper model and/or the prediction of future values based on past observations is desired.

For several decades, time series analysis has been approached by means of stochastic models Box et al. (2015); Pankratz (1983), artificial neural networks (ANNs) Zhang et al. (1998); Adya and Collopy (1998); Zhang (2012), and support vector machines (SVMs) based models Cortes and Vapnik (1995); Van Gestel et al.
One of the most classical approaches to represent linear time series is the auto-regressive moving average (ARMA) model, which combines the concept of auto-regressive (AR) and moving-average (MA) models. However, in practice, many time series have a non-stationary behavior in nature, and ARMA models are inadequate to properly describe this behavior. As an alternative, the auto-regressive integrated moving average (ARIMA) model and its variations have been proposed for non-stationary data as well as to detect certain trend and seasonal patterns. In order to capture non-linear characteristics since linear models are insufficient in many real applications, several methods have been developed such as: the non-linear moving average model (Robinson 1977) and the class of auto-regressive conditional heteroskedasticity (ARCH) models, introduced by Engle (1982). Non-linear models are appropriate for predicting volatility changes in financial time series (Tsay 2010).

SVMs solve pattern classification problems by building maximum margin hyperplanes. They can solve non-linear problems by applying the kernel trick to calculate inner products in a feature space (Aizerman et al. 1964). In Drucker et al. (1997), the concept of SVM was extended to encompass regression analysis, and then other techniques have been developed based on this extension, such as least-squares SVM (LS-SVM) (Suykens and Vandewalle 1999) and Bayesian SVM (Polson and Scott).

ANNs, originally developed to model basic biological neural systems, have being applied in a wide variety of tasks, such as image recognition, data mining, classification, regression analysis, among others. Due to their property of universal approximation (Hornik et al. 1989, 1990; Hornik 1991) and no need to make any a priori assumption about the statistical distribution of the data, ANN has become a powerful tool. Hence, in the last decades, a wide range of applications for time series analysis and forecasting has been solved by neural networks (NNs) (Zhang et al. 1998; Zhang 2012). The most widely used ANNs for regression analysis are multilayer perceptrons (MLPs) with non-linear activation functions, which are composed of an input layer, one or more hidden layers, and the output layers of nodes.

In ANNs, non-linearity is commonly introduced by activation functions for modeling outputs of intermediate and/or final layers with the aim of computing more complex problems, which is valuable for most of ANN applications. According to Cybenko (1989), two-layer NNs with a non-linear function can be proven to be a universal function approximator. This function is usually selected according to some heuristic rules or desired properties, some of them are: rectified linear unit (or ReLU, more used in convolution networks), softmax (used in multi-classification methods), logistic sigmoid (used in binary predictions), hyperbolic tangent (or Tanh) functions, among other variations. Tanh function is just a scaled and shifted version of the logistic sigmoid function but, in addition to that, it is an anti-symmetric function. Non-symmetric functions as sigmoid tend to introduce a source of systematic bias which results in getting stuck during training. Therefore, Tanh function is a more convenient alternative for overcoming this problem and also yielding a faster convergence than non-symmetric activation functions (Haykin 1998).

There is no algorithm for obtaining the global optimal solution of a general non-linear optimization problem in a reasonable amount of time. Besides that, MLPs are usually trained by means of the standard error back-propagation algorithm, which is based on the well-known gradient-descent (GD) algorithm. The error with respect to the desired response is propagated through the network and allows the adjustment of network weights.

Deep neural networks (DNNs) are NNs with a certain level of complexity, which can be described as ANNs with an expressive number of layers connecting input and output (Goodfellow and Courville 2016). Deep learning neural networks have attracted the attention of the machine learning community because of their appealing data-driven framework and of their remarkable performance in a number of pattern recognition tasks. It is well-known that state-of-the-art DNNs are highly redundant and contain hundreds of millions of parameters, using up to all available memory of personal computers. However, attempts to decrease the width and depth of the neural network layers usually lead to considerable drop of performance.

To overcome the limitations inherent to modern DNNs, there is a need for the development of new fast learning algorithms and the application of special data formats for storing the parameters of such network. Current advances in NNs in most cases are associated with heuristic construction of the network architecture and applicable only to a particular problem. On the other hand, there is no understanding of the internal modus operandi of the network, of the necessity or redundancy of certain layers, of the optimal methods to choose hyper-parameters, among others. The lack of comprehensive scientific answers to the above questions
essentially limits the qualitative development of the neural networks method. Furthermore, it is important
to reduce the computing requirements of modern DNNs and a very promising strategy is based on tensor
networks (TNs) Cichocki (2014).

The tensor network (TN) Cichocki (2014) generally decomposes higher-order tensors into sparsely
interconnected matrices or lower-order tensors, and is related to the concept of compression. Tensor
networks are one of the most successful tools in quantum information theory, and are a way of presenting
of multi-dimensional arrays with a small number of parameters. Two special cases of TN are Tensor-Train
(TT) Oseledets (2011); Oseledets and Tyrtyshnikov (2009) and Quantized Tensor-Train (QTT) Oseledets
(2010) decompositions. The TT network is the simplest tensor network, while QTT is a version of TT for
problems of small dimension (vectors and matrices) by introducing virtual dimensions.

DNNs allows learning simultaneously multiple related tasks and using the information learned from
each task in the other layers. This is known as cross-task sharing structure. Nonetheless it is not
simple to determine or to specify the number of tasks per layer and the appropriate sharing structure.
Multidimensional data can be naturally constructed by stacking parameters regarding many tasks in a
higher-order tensor. In Yang and Hospedales (2016), the authors have proposed a generalization of shallow
multi-task learning (MTL) methods to the deep network context with the purpose of applying tensor
factorization of DNN for multi-task representation.

Thus their deep multi-task representation is trained via standard back-propagation. The rank of each
layer is set based on a single task learning initialization. As in Kumar and III (2012), this framework is not
sensitive to rank choice as long as it is large enough. Consequently, their method automatically learns the
shared representation parameters across the tasks, thereby significantly reducing the space of DNN design
choices and not requiring user trial and error.

In Novikov et al. (2015) the authors investigated perspectives of application of the QTT decomposition
for the weights matrix of fully connected layer of DNN. A new formulation of stochastic gradient descent
method was developed for training the network in TT-format using standard approaches. Preliminary results
prove the ability to compress the fully connected layer of ultra DNN by more than 200,000 times.

To tackle the issues associated to ANNs and thanks to the advantages of tensor approaches and its
recent results, the present paper is focused on using TT networks to construct a compact representation
of ANNs and to directly learn the coefficients of TT network based on a given data with the aim of solving a
regression problem. The prediction of times series has played an important role in many science fields of
practical application as engineering, biology, physics, meteorology, etc. In our work we have considered two
different scenarios: noisy chaotic time series, by means of Mackey-Glass equation, and a real financial time
series, given by NASDAQ index.

The Mackey–Glass system has been introduced as a model of white blood cell production Mackey and
Glass (1977), which is usually modeled by delay-differential equations. Diseases can be associated to unstable
oscillations observed in complex mathematical models of human physiological systems. Mackey-Glass
equation provides a range of periodic and chaotic dynamics, which allows to represent the dynamic in
human physiology. Due to the dynamic properties and the mathematical simplicity, the Mackey-Glass time
series has been employed to validate prediction methods, through the forecast of chaotic time series Chng
et al. (1996); Henry Leung et al. (2001); Gu and Wang (2007); Mirzaee (2009); Ko et al. (2011).

Because of the ability to perform complex non-linear modeling, ANNs have been employed in financial
applications in order to predict markets’ behavior and as well as to provide a powerful statistical modeling
 technique that can be an alternative to traditional methodologies, such as ARMA, ARCH and Generalized
ARCH (GARCH) models Tang et al. (1991); Gately (1995); Chatterjee et al. (2000). In this sense, we have
considered the NASDAQ index forecasting in order to study the learned model in a real case.

This paper is organized as follows. Sections 2 describes and discusses the learning model based on
TT networks, by proposing a reduction of computational calculus and by deriving a regularization matrix
factor. Section 3 analyses the optimization framework and discusses an alternative strategy to reduce the
computational cost of pseudo-inverse calculus. Section 4 discusses some general considerations regarding
tensor and neural networks. In Section 5, a comparative analysis is carried out in the context of neural
network recovery and non-linear predictions of two time series. Finally, Section 6 presents some conclusions.
Notations and property: Scalars, column vectors, matrices, and higher-order tensors are written with lower-case, boldface lower-case, boldface upper-case, and calligraphic letters, i.e. \((a,\mathbf{a},\mathbf{A},\mathcal{A})\), respectively. \(\mathbf{A}^T\) and \(\mathbf{A}^{-1}\) stand for transpose and inverse matrices of \(\mathbf{A}\), respectively. \(\mathbf{0}_N\) stands for a column vector consisting of \(N\) zero, \(\mathbf{I}_N\) is the identity matrix of order \(N\), \(\mathcal{N}(\cdot)\) denotes a null-space of a matrix, \(\|\cdot\|_2\) is the Euclidean norm, whereas \(\langle\cdot\rangle\) denotes the scalar product of two tensors (extension of the classical scalar product between two vectors). The operators vec(\(\cdot\)) and unfold(\(\cdot\)) form a column vector by stacking the columns of its tensor argument and a matrix by arranging the modes of an input tensor, respectively. The \(n\)-mode product of a tensor and vector, defined as \(\mathcal{A} \times_n \mathbf{x}\), represents a contraction of tensor to a low-order tensor. The Kronecker and Khatri-Rao products are denoted by \(\otimes\) and \(\diamond\), respectively. Given \(\mathbf{X} \in \mathbb{C}^{I \times K}\) and \(\mathbf{Y} \in \mathbb{C}^{J \times K}\), both Kronecker and Khatri-Rao products are related according with \(\mathbf{X} \diamond \mathbf{Y} = [x_1 \otimes y_1, \ldots, x_K \otimes y_K] \in \mathbb{C}^{IJ \times K}\), where both vectors \(x_k\) and \(y_k\) denote the \(k\)-th column of \(\mathbf{X}\) and \(\mathbf{Y}\). A useful Kronecker property is given by
\[
\text{vec}(\mathbf{ABC}) = (\mathbf{C}^T \otimes \mathbf{A}) \text{vec}(\mathbf{B}).
\]

2. Learning of predictive model

In supervised machine learning, given a training dataset of pairs \(\{\mathbf{x}^{(m)}, y^{(m)}\}\), for \(m \in \{1, \ldots, M\}\), where each input vector \(\mathbf{x}^{(m)}\) is associated with a desired output \(y^{(m)}\), the target output can be predicted according to the following model:
\[
\hat{y}^{(m)} = \langle \mathcal{W}, \phi(\mathbf{x}^{(m)}) \rangle = \mathbf{W} \times_1 \phi(x_1^{(m)}) \cdots \times_N \phi(x_N^{(m)}),
\]
where each \(m\)-th input vector \(\mathbf{x}^{(m)} \doteq [x_1^{(m)}, \ldots, x_N^{(m)}]^T \in \mathbb{R}^N\) is mapped onto a higher-order dimensional space through a feature map \(\phi: \mathbb{R}^N \rightarrow \mathbb{R}^{S_1 \times \cdots \times S_N}\). The parameter \(\mathcal{W}\) determines how each feature affects the prediction.

The most common method used for fitting regression problems is based on the least squares (LS) method. Thus, the predictors resulting from this model, i.e., those based on \(\mathcal{W}\), can be learned by minimizing the mean squared error (MSE) function:
\[
l(\mathcal{W}) = \frac{1}{M} \sum_{m=1}^M \left(\langle \mathcal{W}, \phi(\mathbf{x}^{(m)}) \rangle - y^{(m)}\right)^2 = \frac{1}{M} \|\hat{y} - \mathbf{y}\|^2_2,
\]
where \(\mathbf{y} \doteq [y^{(1)}, \ldots, y^{(M)}]^T \in \mathbb{R}^M\) and \(\hat{y} \doteq [\hat{y}^{(1)}, \ldots, \hat{y}^{(M)}]^T \in \mathbb{R}^M\) denote respectively the concatenation of all desired outputs and its predictions associated with the input vectors \(\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}\}\).

Feature functions, as well as the weighting tensor, can be exponentially large. In our case, both \(N\)-th order tensors \(\mathcal{W}\) and \(\phi\) have \(S_1 S_2 \cdots S_N\) components. An interesting way to reduce the number of coefficients is regarding \(\mathcal{W}\) as a particular structure, such as Tensor-Train (TT) decomposition Oseledets (2011), since it is given by a sequence of low-order tensors in accordance with
\[
\mathcal{W}_{s_1, s_2, \ldots, s_N} = \sum_{r_1, \ldots, r_N = 1} \mathcal{G}^{(1)}_{r_0, s_1, r_1} \cdots \mathcal{G}^{(N)}_{r_N-1, s_N, r_N}
= \sum_{r_1, \ldots, r_N = 1} \prod_{n=1}^N \mathcal{G}^{(n)}_{r_n-1, s_n, r_n},
\]
where each tensor, called TT-core, is denoted by \(\mathcal{G}^{(n)} \in \mathbb{R}^{R_{n-1} \times S_n \times R_n}\) for all \(n \in \{1, \ldots, N\}\) with \(r_n \in \{1, \ldots, R_n\}\), \(s_n \in \{1, \ldots, S_n\}\), and \(R_0 = R_N = 1\).
isolating the decomposition, thereby, in general, the TT-rank is constrained by
where both vectors $p$ and $n$ can be expressed in terms of the $R_k$ denoting an unfolded matrix of
+1
k
m
$k$

+2
k
+1

+1
k

$G^{(n)}$ $\times_n \phi(x_n^{(m)})$
$= G^{(k)} \times_1 p_{k-1}^{-}(x^{(m)}) \times_2 \phi(x_k^{(m)}) \times_3 p_{k+1}^{+}(x^{(m)})$
$= \left( p_{k-1}^{-}(x^{(m)}) \otimes \phi(x_k^{(m)}) \otimes p_{k+1}^{+}(x^{(m)}) \right) \text{vec}(G^{(k)})$,
where both vectors $p_{k-1}^{-}(x^{(m)})$ and $p_{k+1}^{+}(x^{(m)})$ represent respectively the contraction of the left and right sides of the TT structure, i.e.
\[
\begin{cases}
    p_{k-1}^{-}(x^{(m)}) \triangleq \prod_{n=1}^{k-1} G^{(n)} \times_n \phi(x_n^{(m)}) \in \mathbb{R}^{R_{k-1}} \\
    p_{k+1}^{+}(x^{(m)}) \triangleq \prod_{n=k+1}^{N} G^{(n)} \times_n \phi(x_n^{(m)}) \in \mathbb{R}^{R_k}
\end{cases}
\]
Observe that both vectors $p_{k-1}^{-}(x^{(m)})$ and $p_{k+1}^{+}(x^{(m)})$ can be rewritten as a function of the previous value, i.e.
\[
\begin{cases}
    p_{k-1}^{-}(x^{(m)}) = G^{(k-1)} \times_1 p_{k-2}^{-}(x^{(m)}) \times_2 \phi(x_{k-1}^{(m)}) \\
    p_{k+1}^{+}(x^{(m)}) = G^{(k+1)} \times_1 \phi(x_{k+1}^{(m)}) \times_3 p_{k+2}^{+}(x^{(m)})
\end{cases}
\]
Thus by sweeping from left-to-right (or right-to-left), we can use $p_{k-2}^{-}(x^{(m)})$ (or $p_{k+2}^{+}(x^{(m)})$) to compute $p_{k-1}^{-}(x^{(m)})$ (or $p_{k+1}^{+}(x^{(m)})$) respectively. Using (7) instead of (6) will reduce the demanding computational operations per each k-th core estimation, in terms of complex multiplications, once we can use the previous calculus of $p_{k-2}^{-}(x^{(m)})$ (or $p_{k+2}^{+}(x^{(m)})$), which leads to $(S_{k-1} + 1) R_{k-2} R_{k-1}$ (or $(S_{k+1} + 1) R_k R_{k+1}$) instead of $\sum_{n=1}^{k-1} R_{n-1} S_n R_n + \sum_{n=2}^{k-1} R_{n-2} R_{n=1} R_n$ (or $\sum_{n=k+1}^{N} R_{n-1} S_n R_n + \sum_{n=k+1}^{N-1} R_{n-1} R_n R_{n+1}$) respectively.

From the concatenation of all outputs \{$\hat{y}^{(1)}, \ldots, \hat{y}^{(M)}$\}, and by applying (5), the estimated vector of the desired vector $y$ can be expressed in terms of the $k$-th core $G^{(k)} \in \mathbb{R}^{R_{k-1} \times S_k \times R_k}$, i.e. $\theta_k$, by
\[
y = (\Phi_k \circ P_{k-1}^{-} \circ P_{k+1}^{+})^T \text{vec}(G_{R_{k-1}R_kS_k}^{(k)}) = P_k \theta_k \in \mathbb{R}^M,
\]
where $G_{R_{k-1}R_kS_k}^{(k)}$ denotes an unfolded matrix of $G^{(k)}$ and
\[
\begin{cases}
    \Phi_k \triangleq \left[ \phi(x_k^{(1)}) \cdots \phi(x_k^{(M)}) \right] \in \mathbb{R}^{S_k \times M} \\
    P_{k-1}^{-} \triangleq \left[ p_{k-1}^{-}(x^{(1)}) \cdots p_{k-1}^{-}(x^{(M)}) \right] \in \mathbb{R}^{R_{k-1} \times M} \\
    P_{k+1}^{+} \triangleq \left[ p_{k+1}^{+}(x^{(1)}) \cdots p_{k+1}^{+}(x^{(M)}) \right] \in \mathbb{R}^{R_k \times M} \\
    P_k \triangleq (\Phi_k \circ P_{k-1}^{-} \circ P_{k+1}^{+})^T \in \mathbb{R}^{M \times S_k \times R_{k-1} R_k} \\
    \theta_k \triangleq \text{vec}(G_{R_{k-1}R_kS_k}^{(k)}) \in \mathbb{R}^{S_k R_{k-1} R_k}
\end{cases}
\]
Remark that the remaining core tensors are absorbed in the matrix $P_k$, from the above manipulations in accordance with (5)-(7), and the $k$-the core tensor $G^{(k)}$ is isolated in the expression (8) with the aim of rewriting the loss function (3) in terms of the $k$-the core tensor. The importance of this procedure will become clearer in the next section.

Finally, the loss function, given in (3), can be also expressed in terms of both vectors $y$ and $\hat{y}$, respectively associated with all target outputs and its predictions, by applying (8) to (3) in the form

$$l(W) = \frac{1}{M} \|P_k \theta_k - y\|_2^2.$$ (10)

If $P_k$ has linearly independent columns, then $P_k^T P_k$ is non-singular and the solution of least squares regression given by (10) turns out

$$\frac{\partial}{\partial \theta_k} l(W) = 2 \frac{1}{M} (P_k^T P_k \theta_k - P_k^T y)$$

$$\hat{\theta}_k = (P_k^T P_k)^{-1} P_k^T y$$ (11)

where $\hat{\theta}_k \Delta \text{vec}(\tilde{G}^{(k)}_{R_{k-1} \times R_k \times S_k} - \Delta)$ denotes an estimate of $\theta_k$ and, consequently, an estimate of $G^{(k)}$.

### 2.1. Shrinkage regularization method

The colinearity (or multicolinearity) phenomenon affects calculations regarding individual predictors, in the sense that one predictor can be linearly determined through the others with a substantial degree of accuracy which leads to an inversion problem due to rank deficient of $P_k$. In order to ensure that $P_k^T P_k$ is not ill-conditioned due to correlated columns of $P_k$, i.e. collinear rows of $P_k$ and $P_k^+$ owing to Khatri-Rao structure given by (9), we can consider a regularization term $r(W)$ added to the loss function (10). Thus, we are minimizing the following function:

$$l'(W) = l(W) + \lambda r(W),$$ (12)

where $\lambda \geq 0$ denotes the regularization or shrinkage factor.

One common option, initially motivated to stabilize the solution (11), is based on the $l_2$-norm of the weighting coefficients, also referred to as Tikhonov regularization Kendall et al. (1991). In statistical literature, it is also known as ridge regression Hoerl and Kennard (1970) and the regularization term can be given by

$$r(W) = \langle W, W \rangle = \|W\|^2_F.$$ (13)

In order to obtain a regularization expression in terms of $\theta_k$, regarding a TT-format for weighting tensor $W$ in (4), we can rewrite the scalar product in (13) by isolating the $k$-th core $G^{(k)}$ and contracting recursively the remaining cores $1 \leq n \leq k-1$ and $k + 1 \leq n \leq N$ as follows

$$\tilde{G}^{(k-1)^-} \Delta \text{unfold} \left( \prod_{n=2}^{k-1} G^{(n)}_1 \times_1 \tilde{G}^{(n-1)^-} \right),$$

$$\in \mathbb{R}^{S_1 \cdots S_{k-1} \times R_{k-1}}$$

$$\tilde{G}^{(k+1)^+} \Delta \text{unfold} \left( \prod_{n=k+1}^{N-1} G^{(n)}_3 \times_3 \tilde{G}^{(n+1)^+} \right)^T,$$

$$\in \mathbb{R}^{S_{k+1} \cdots S_N \times R_k}$$ (14)
where \( \tilde{G}^{(n)-} \in \mathbb{R}^{S_1 \cdots S_n \times R_n} \) and \( \tilde{G}^{(n)+} \in \mathbb{R}^{S_n \cdots S_N \times R_{n-1}} \) are recursively obtained according to

\[
\tilde{G}^{(n)-} \triangleq \begin{cases} 
G^{(1)} \in \mathbb{R}^{S_1 \times R_1}, & n = 1 \\
\text{unfold}(\tilde{G}^{(n)} \times_1 \tilde{G}^{(n-1)-}), & 2 \leq n \leq k - 1 \\
\text{unfold}(G^{(n)} \times_3 \tilde{G}^{(n+1)+})^T, & k + 1 \leq n \leq N - 1 \\
G^{(N)T} \in \mathbb{R}^{S_N \times R_{N-1}}, & n = N
\end{cases}
\]  

(15)

with \( \tilde{A} \triangleq \text{unfold}(A) \) defining the unfolding operation of a tensor \( A \in \mathbb{R}^{a_0 \times a_1 \times a_2} \) into a matrix \( \tilde{A} \in \mathbb{R}^{a_0 a_1 \times a_2} \), by contracting the two first modes in one mode.

Finally, we can represent the tensor of coefficients \( W \), defined in (4), from (14) by means of an unfolded matrix as follow

\[
\text{unfold}(W) = \left( \tilde{G}^{(k-1)-} \otimes \tilde{G}^{(k+1)+} \right) G^{(k)}_{R_{k-1} \times R_k \times S_k} \\
= \prod_{n=1}^{k} S_n \times S_k \in \mathbb{R}^{n \times \# k}.
\]  

(16)

The vectorization of a higher-order tensor can be derived from the vectorization of a matrix unfolding of this tensor. Observe that the order of the dimensions is quite relevant because it denotes the speed at which each mode changes.

By applying the Kronecker property (1), we can represent the above matrix as a vector given by

\[
\text{vec}(W) = L_k \text{vec}\left( G^{(k)}_{R_{k-1} \times R_k \times S_k} \right) = L_k \theta_k \in \mathbb{R}^{S_k \prod_{n=1}^{N} S_n \times R_{n \times \# k}}, \text{ with}
\]

\[
L_k \triangleq I_{S_k} \otimes \left( \tilde{G}^{(k-1)-} \otimes \tilde{G}^{(k+1)+} \right) \in \mathbb{R}^{S_k \prod_{n=1}^{N} S_n \times S_{k-1} \times R_k}.
\]  

(17)

From (13)-(17), we can write the regularization term as a function of the \( k \)-th core \( \tilde{G}^{(k)} \), given by \( \theta_k \), according to

\[
r(W) = \| \text{unfold}(W) \|_F^2 = \| \text{vec}(W) \|_F^2,
\]

(18)

and the gradient vector with respect to \( \theta_k \) is

\[
\frac{\partial}{\partial \theta_k} r(W) = 2 L_k^T L_k \theta_k = 2 \left( I_{S_k} \otimes B_k^T B_k \right) \theta_k.
\]  

(19)

Regarding the linear LS problem based on the loss function (12), i.e.

\[
\text{minimize} \quad \frac{1}{2m} \| P_k \theta_k - y \|_2^2 + \lambda \| L_k \theta_k \|_2^2,
\]

(20)

and under the assumption that the null-spaces of \( P_k \) and \( L_k \) intersect only trivially, i.e.

\[
N(P_k) \cap N(L_k) = \{0\} \iff \text{rank} \left( \begin{bmatrix} P_k \\ L_k \end{bmatrix} \right) = S_k R_{k-1} R_k,
\]  

(21)
the LS problem (20) has the unique solution for any $\lambda > 0$ given by F. van Loan (1976); Eldén (1977); Hansen (1989)

$$
\frac{\partial}{\partial \theta_k} l(W) = \left( \frac{2}{M} P_k^T P_k + 2\lambda L_k^T L_k \right) \theta_k - \frac{2}{M} P_k^T y,
$$

$$
(P_k^T P_k + \lambda M L_k^T L_k) \theta_k = P_k^T y \tag{22a}
$$

$$
\hat{\theta}_k = (P_k^T P_k + \lambda M L_k^T L_k)^{-1} P_k^T y. \tag{22b}
$$

In case the condition (21) is not met, the solution (22b) is not unique.

Remark that this regularization term penalizes large values of weighting coefficients for $\lambda > 0$ and becomes just a linear regression for $\lambda = 0$ (no regularization). For $\lambda > 0$, it makes the problem nonsingular, as the matrix we need to invert no longer has a determinant near zero in the sense that its eigenvalues are no longer near zero, which avoids imprecise estimation of the inverse matrix Kendall et al. (1991). Moreover, analogously to principal components analysis (PCA), ridge regression suppresses the low-order principal components, since the parameter $\lambda$ greatly affects the smallest singular values (SV) of $P_k$ in contrast to the largest SV, which are only slightly affected. In case $\lambda$ is chosen too large, the problem we solve has no longer connection to the original problem.

Besides solving ill-posed optimization problems, the use of regularization with an appropriate weight decay can also prevent over-fitting problem, by reducing model complexity, through model constraints on the parameter $\theta_k$ in order to enhance the compromise between prediction accuracy i.e. having a small residual error, and robustness and model flexibility. Meanwhile, a very large $\lambda$ leads to under-fitting, once it forces the model to represent a constant loss function, i.e. the model becomes too simple to learn the underlying structure of data Ian Goodfellow and Courville (2016). Therefore, the adjust of $\lambda$ controls the effective degrees of freedom of the model Hastie et al. (2009), i.e. the model's capacity.

Two other common shrinkage methods are: Lasso (Least Absolute Shrinkage and Selection Operator) regression Tibshirani (1996) i.e. $r(W) = ||W||_1$, which induces sparsity constraint by replacing the $l_2$-norm with the $l_1$-norm, and Elastic net Zou and Hastie (2005) i.e. $r(W) = \alpha ||W||_1 + (1 - \alpha) ||W||_2^2$ for $\alpha \in [0, 1]$, designed to overcome limitations of Lasso (convex problem but not strictly differentiable), allows across $\alpha$ the control of a mix between the regularization terms of Ridge (for $\alpha = 0$) and Lasso (for $\alpha = 1$) Hastie et al. (2009).

In practical terms, Lasso regression tends to perform feature selection by setting weights near to zero, which leads to ignore the least important or even irrelevant information from datasets; whereas Elastic net is preferred when several features are strongly correlated. Additionally, in case that the number of observations $M$ is greater than the number of model parameters $P$, ridge regression tends to perform better than Elastic net Zou and Hastie (2005).

There are several variants of Lasso penalty developed to tackle certain optimization limitations and to address to particular problems Hastie et al. (2009). Furthermore, another alternative is to constraint norm penalties by inequality functions given by a constant $k$, i.e. $r(W) > k$ or $r(W) < k$, which can be dealt with using the Karush-Kuhn-Tucker conditions, an extension of the solution via Lagrange multipliers, applicable only for equality constraints. However, to analytically derive a closed-form solution is not recurrently possible since the penalties can give rise to a non-convex optimization problem (and, consequently, to convergence to local minima), besides determining an appropriate constant $k$ Ian Goodfellow and Courville (2016).

2.2. Feature map: Encoding input data

In machine learning, feature maps can be specified in accordance with certain learning tasks in order to exploit the correlation of information inherent into input data and better classify or estimate it. Thus, input data could implicitly encode a localization information with the purpose of associating set of pixels to detect more efficiently a particular object in an image for example. Furthermore, feature mapping can allow non-linearly separable data to become linearly separable by a hyper-plane in a higher-order dimension.

According to (2), the same local feature, defined by $\phi(\cdot): \mathbb{R} \rightarrow \mathbb{R}^S$, is applied to each input $x_n^{(m)}$. Fitting a linear regression model may not be adequate when interactions between variables are not inherently linear.
However, the linear regression framework can still be used if the model is nonlinear but linear with respect to its parameters. This is possible by means of a transformation applied to each input, such as a power or logarithmic transformation for example. We can include logarithmic transformation of features by regarding exponential regression model. As an example, for a three-dimension array, $S_n = 3$, we have

$$\phi \left( x_n^{(m)} \right) = \left[ 1 \ x_n^{(m)} \ \log(x_n^{(m)}) \right]^T.$$  \hspace{1cm} (23)

Another possible way of generating nonlinear interaction features is to consider a polynomial regression model of degree $S_n - 1$, which can be expressed by the Vandermonde structure (for $S_n = 3$) given by

$$\phi \left( x_n^{(m)} \right) = \left[ 1 \ x_n^{(m)} \ x_n^{(m)2} \right]^T.$$  \hspace{1cm} (24)

Note that the first-order polynomial leads to a multiple linear model whereas higher order ($S_n \geq 3$) allows a better fit for polynomial curves.

Remark that, in our approach, each TT-core $G^{(m)} \in \mathbb{R}^{R_{n-1} \times S_n \times R_n}$ is used for mapping the existing interactions between inputs per each categorical feature. Therefore, the number of cores is determined by the number of features for a given data and the feature map regards the structure of inputs by exploiting nonlinear relationships.

3. Optimization framework

To design a learning algorithm, the parameters of our model can be derived from minimizing the mean of squared residuals on the training set under the TT-rank constraint. From (3), it leads to

$$\text{minimize} \quad W \quad \frac{1}{M} \sum_{m=1}^{M} \left( \langle W, \phi(x_n^{(m)}) \rangle - y_n^{(m)} \right)^2.$$  \hspace{1cm} (25)

subject to $\text{TT-rank} = R$.

Several algorithms have been proposed to approximate TT-tensor by applying well-known approaches to solve (25), such that gradient descent, conjugate gradient, and Riemannian optimization methods Stoudenmire and Schwab (2016); Novikov et al. (2016). However, a weak point of some of these approaches is due to the estimation of two consecutive cores at time. After estimating a contraction of both cores $k$ and $k+1$, a restoring procedure based on the approximation by means of SV or QR decomposition is employed. The performance and convergence behavior of these algorithms are strongly dependent upon the adopted approximation, which is also limited by TT-rank.

An alternative strategy is to convert the optimization problem (25) into independent linear least squares problems for adaptively estimating only one core tensor $G^{(k)}$ at a time by sweeping along all core tensors from left-to-right and right-to-left, by fixing the remaining cores. According to the development made in Section 2, we can rewrite the overall problem (25) with a regularization factor by using (20) as the following optimization approach

$$\text{minimize} \quad \theta_1, \ldots, \theta_N \quad \left\{ \sum_{k=1}^{N} ||P_k \theta_k - y||_2^2 + \lambda M \|L_k \theta_k\|_2^2 \right\}.$$  \hspace{1cm} (26)

subject to $\text{TT-rank} = R$.

To reduce the computational complexity effort required for evaluating the solution in (22b) for several values of $\lambda$, we can first apply the Generalized Singular Value Decomposition (GSVD) of the matrix pair $(P_k, L_k)$, proposed by Van Loan F. van Loan (1976) assuming $M \geq S_k R_{k-1} R_k$ and the condition in (21), which is given by

$$\begin{cases} P_k = U \Sigma_P X^T \\ L_k = V \Sigma_L X^T \end{cases}$$  \hspace{1cm} (27)
where \( U \) and \( V \) are orthogonal matrices, \( \Sigma_P \) and \( \Sigma_L \) are diagonal matrices and \( X \) is non-singular matrix.

By replacing (27) in (22a), it gives an equivalent minimization problem, after some manipulations regarding \( z_k \approx X^T \theta_k \), it gets

\[
(\Sigma_P^T \Sigma_P + \lambda M \Sigma_L^T \Sigma_L) z_k = \Sigma_P^T U^T y
\]

\[
\hat{z}_k = (\Sigma_P^T \Sigma_P + \lambda M \Sigma_L^T \Sigma_L)^{-1} \Sigma_P^T U^T y,
\]

(28a)

\[
\hat{\theta}_k = X^{T^{-1}} z_k.
\]

(28b)

From (28a), the inverse calculation is reduced to the inverse of each element on the diagonal, the decomposition in (27) and the inverse matrix in (28b) are computed just once for several values of \( \lambda \).

There are different approaches to compute the GSVD or based on the GSVD, such as those discussed in Eldén (1982); Hansen (1989); Golub and Van Loan (1996); Morigi et al. (2007); Dykes and Reichel (2014), with the aim of reducing the computational effort and/or exploiting the structure of the regularization matrix. In Eldén (1982); Morigi et al. (2007), the GSVD computations take advantage of the structure of the regularization matrix, in case it is a band matrix or an orthogonal projection operator respectively. Additionally, Eldén in Eldén (1982) discussed an alternative way to solve (20), in case \( L_k \) is not square and invertible, by considering a weighted inverse matrix which allowed the transformation of the original problem to a standard-form problem. Unlike those cases, the paper Dykes and Reichel (2014) proposed, based on Eldén (1982), a method for computation of the GSVD and the truncated GSVD (TGSVD), proposed by Hansen in Hansen (1989) which generalizes truncated SVD, when the regularization matrix does not have an exploitable structure. Furthermore, Dykes and Reichel presented in Dykes and Reichel (2014) an approach for reducing the matrix pair \((P_k, L_k)\) to a a pair of simpler matrices in order to reduce the GSVD computations.

Remark that our regularization matrix \( L_k \), defined in (17), is a Kronecker product between \( I_{S_k} \) and \( B_k \). Therefore, it is a band matrix that enables to exploit the sparseness of its structure in the numerical computation regarding the regularization matrix, in accordance with the approaches discussed in Eldén (1977); Eldén (1982); Björck (1988). This analysis was not included in the scope of our study once there are several works proposed on this topic, as commented below.

The algorithmic details of our proposed technique for multilinear regression model is presented in Algorithm 1. Note that the estimation of each TT-core is conditioned by the knowledge of previous estimating cores and an intermediate orthogonalization step is included by the QR decomposition, applied to each reshaped TT-core tensor defined in steps 8 (Algorithm 1), with the aim of guaranteeing the left and right orthogonality property of TT cores and consequently, the algorithm stability Savostyanov and Oseledec (2011); Oseledec (2011). The matrix \( \hat{Q} \) in the step 9 is built taking into account the \( r \) first columns of \( Q \), such that \( r = \min(\text{rank}(\hat{G}^{(k)}_{R_k \times S_k \times X_k}), \hat{R}) \). The criteria for selecting \( \lambda \) is detailed in the next sections.

Remark that each core estimation problem can be seen as a layer in the network model, from which inputs with information \( x^{(m)} \), \( \forall m \in \{1, ..., M\} \), flow forward through the network. Hence the estimation of each core propagates the initial information along all network taking into account one feature per layer and finally produces the desired output. During the training, the sweeping procedure, widely applied for approximating TT structure, also allows that the information flow backwards through the network. Thus it can be analogously associated with the back-propagation learning in artificial neural network.

4. General considerations

In regression analysis, it is quite usual to standardize the inputs before solving (26) i.e. reparametrization using centered inputs, in order to avoid multicollinearity issues, which could affect model convergence, and also meaningful interpretability of regression coefficients. Consequently, it leads to estimate coefficients of ridge regression model without intercept Hastie et al. (2009).

The learning procedure of network weights is derived by the minimization of loss function, which is chosen based on a criterion to approximate the model, leading to a direct approximation between the desired output
Algorithm 1 TT-MR: Multilinear regression model

1: Random initialize all cores \{G^{(1)}, \ldots, G^{(N)}\}
2: Compute encoded inputs \{\Phi_1, \ldots, \Phi_N\}
3: while stop condition is not met
4: for \(k = 1, 2, \ldots, N\)
5: \quad Compute \(P_{k-1}^-\) and \(P_{k+1}^+\) using (6) and (9)
6: \quad Select \(\lambda\) according to the lowest cost function
7: \quad Compute \(\tilde{\theta}_k \triangleq \text{vec}(\tilde{G}^{(k)})\) by solving (22a) or (28b)
8: \quad Compute QR decomposition: \(\tilde{G}^{(k)}_{R_k-S_k \times R_k} = QR\)
9: \quad Update \(\tilde{G}^{(k)}\) from \(\text{vec}(\tilde{G}^{(k)}) \leftarrow \text{vec}(\tilde{Q})\)
10: Change sweeping order
11: return \(W\) in TT-format with cores \{G^{(1)}, \ldots, G^{(N)}\}

and its prediction. Mean squared error has been classically used as loss function in the context of regression problems Bishop (2006); Haykin (2013). Depending on the model, it is not possible to derive a direct and closed expression for the optimal coefficients. In this sense, several iterative methods has been proposed to solve it based on the well-known GD optimization technique.

There are several versions that aim to accelerate the standard GD method, such that Stochastic Gradient Descent (SGD) with momentum method Qian (1999) (or Momentum, which helps accelerate SGD), Nesterov Accelerated Gradient Nesterov (1983) (or simply NAG, variant of Momentum), Adaptive Gradient Duchi et al. (2011) (or Adagrad, learning rate for each parameter according to the history of the gradients for that parameter), RMSProp Tieleman and Hinton (2012) (very similar to Adagrad but the update step is normalized by a decaying Root-Mean-Square-RMS of recent gradients), Adadelta (it tends to be more robust to the choice of learning rate of Adagrad), Adaptive Momentum Estimation Kingma and Ba (2014) (or Adam, it takes advantage of both RMSProp and Adadelta by including adaptive momentum), among many others. Adam and RMSProp are two very popular optimizers still being used in most neural networks. Both update the variables using an exponential decaying average of the gradient and its squared.

In artificial neural networks, the minimization of loss function involves the chain rule across sequential derivatives with respect to the weights of each layer. This procedure is known as error back-propagation through the network. In addition to the algorithmic complexity, the more layers and non-linearities through activation functions are introduced, the more the network becomes complex, increasing the computational effort.

The choice of adaptive learning-method algorithms is dependent on the optimization problem and the method robustness noticeably affects convergence. The focus of this work is mainly to compare tensor and neural networks in terms of their structures, by means of robustness, prediction performance and network complexity. Taking it into consideration, we limit our analysis to the standard GD method and to the Adam algorithm, because its popularity in the domain.

Differently from standard model parameters, hyper-parameters are employed in most machine learning algorithms to control the behavior of learning algorithms and there is no a closed formula to uniquely determine them from data. In general, they are empirically set by searching for the best value by trial and error, such that regularization factor, dropout rate, parameters of optimization algorithm (e.g. learning rate, momentum term, decay rate), among others. A usual way to find the best hyper-parameters is to regard the validation set and a search interval; therefore, this procedure, properly described in Section 5, is equivalently applied to both approaches.

It is usual to evaluate the performance progression, i.e., convergence speed of neural networks in terms of epochs. Every epoch considers the entire data set to update the neural network. In order to set a fair comparison between tensor and neural networks, we take into account the contribution of the entire data on the update of all weights and, in this sense, it is reasonable to put on the same level the algorithmic
convergence according to epochs and sweeps.

5. Simulation Results

A usual way of evaluating model performance is to estimate the weighting tensor learning from a given training data set and to analyze the model performance over a given test data set, which was not used during the learning step. To validate and better understand different aspects regarding the neural and tensor networks, we consider three different experiments described in the following three subsections.

In order to evaluate and compare the performance of the models, we consider the mean squared error (MSE) of predictions, which is given by the loss function, and three other common metrics employed on regression problems: the explained variance score (briefly referred to here as score), which measures the discrepancy between target and its prediction in terms of the sample variance (i.e. the quality of the fit of a model on data), the sample Pearson correlation coefficient (shortly referred to as SPCC), which measures the linear correlation between both variables (target and its prediction) regarding the estimates of co-variances and variances, and the coefficient of determination (known as $R$-squared or $R^2$), which measures the degree of linear correlation and it is unable to determine whether the predictions are biased. These metrics are given by the following expressions:

$$
\begin{align*}
\rho_{\text{MSE}} &= \frac{1}{M} \sum_{m=1}^{M} \left( \hat{y}(m) - \bar{y} \right)^2, \\
\rho_{\text{SCORE}} &= \frac{\text{var}(\hat{y}_{\text{target}})}{\text{var}(\bar{y})} \\
\rho_{\text{SPCC}} &= \frac{\sum_{m=1}^{M} \left( \hat{y}(m) - \bar{y} \right) \left( \hat{y}(m) - \bar{y} \right)}{\sqrt{\sum_{m=1}^{M} \left( \hat{y}(m) - \bar{y} \right)^2 \sum_{m=1}^{M} \left( \hat{y}(m) - \bar{y} \right)^2}}, \\
\rho_{R^2} &= 1 - \frac{\sum_{m=1}^{M} \left( \hat{y}(m) - \bar{y} \right)^2}{\sum_{m=1}^{M} \left( \hat{y}(m) - \bar{y} \right)^2}
\end{align*}
$$

(29)

where $\text{var}(\cdot)$ denotes the sample unbiased variance operator, and $\bar{y}_{\text{target}}$ and $\bar{y}_{\text{target}}$ mean the sample mean of the vector of target $y_{\text{target}}$ and its prediction $\hat{y}_{\text{target}}$.

The weights of tensor and neural networks are only learned from the training and validation sets and the inputs of both networks are scaled to fit the range $[-1, 1]$. It is known that this scaling procedure can provide an improvement on the quality of the solution, as it ensures all inputs are treated equally in the regularization process and allows a meaningful range for the random starting weights Hastie et al. (2009).

The starting values for weights are usually chosen to be random values close to zero. A good practice is to initialize the weights following the uniform distribution in the range of $[-\delta, \delta]$, where $\delta = \frac{1}{\sqrt{n}}$ and $n$ denotes the number of coefficients associated to each neuron, and the biases to be zero. In analogy, the coefficients of each core tensor are also initialized according to this practice, by regarding $n$ in terms of the number of coefficients of each $n$-th core tensor $G^{(n)}$.

The stopping criterion is based on early stopping (in order to avoid overfitting), which is defined as a minimum relative improvement of $10^{-6}$ over, at least, 20% of the maximum number of epochs or sweeps. In all simulations, the data is separated in three different sets for training (60%), validation (20%) and test (20%).

5.1. Recovering multilayer perceptrons

Firstly, we consider a data set with 10000 samples generated by means of a neural network (10-200-1) with 10 inputs and 200 neurons in the hidden layer, totaling 2401 coefficients. The input matrix, randomly generated by a uniform distribution into the range $[-1, 1]$, is propagated in two layers: hidden and output layer. Both weights and biases of the neural network are drawn from a Gaussian distribution with zero-mean and standard deviation equal to 2. Two activation functions, ReLU and $Tanh$ functions, are included in the
intermediate layer in order to introduce a non-linearity in the neural network model. We have considered a maximum number of sweeps equal to 12, since the algorithm convergence is achieved with less number of sweeps.

The weights of tensor networks are only learned from the training and validation sets. The regularization factor $\lambda$ is selected according to a searching step based on the known *Golden-search section* (GSS) with a rough preliminary search regarding the given interval $\{2^n : n \in \mathbb{Z}, -10 \leq n \leq 10\}$. Thus, the optimal regularization factor for each $k$-th core estimate is chosen by taking into account the lowest value of the loss function computed from the validation set.

The neural network output was recovered by the 10-th order Tensor-Train decomposition by fixing a maximum TT-rank ($R$), considering several values, and two values of dimension array ($S = 2$ and $S = 3$), regarding the local feature mapping $\mathbb{R} \to \mathbb{R}^S$ given by the polynomial regression in (24). Tables 1 and 2, for Tanh and ReLU functions respectively, show the average performance for all configurations, over 100 Monte Carlo simulations, in terms of MSE, score, SPCC, and $R$-squared at the convergence, for training, validation and test sets.

According with Table 1, we verify that the performance is improved with the increment of both model parameters $R$ and $S$ once more coefficients are employed. From 232 to 2728 coefficients, for $S = 2$ with $R = 4$ and $R = 40$, we obtained an improvement over the test set of 4.92% in terms of the explained variance score. Analogously for $S = 3$ with $R = 2$ and $R = 12$, from 108 and 2556 coefficients, we got an improvement of 12.53% over the test set. Note that the TT model for $S = 3$ and $R = 14$, with 3288 coefficients, does not provide a better score than the one for $S = 3$ and $R = 12$, with 2556, thus more coefficients lead to a slight over-fitting of the model.

In contrast to the results for recovering the NN with Tanh function, Table 2 shows a lower improvement with the increase of $R$ and $S$. From $R = 20$ with $S = 2$, i.e. from more than 1960 coefficients, the model does not offer a meaningful improvement over the test set, i.e. lower than four decimal places. From 232 to 1960 coefficients, for $S = 2$ with $R = 4$ and $R = 20$, we have a gain over the test set of 1.24% against 10.34% for $S = 3$ with $R = 2$ and $R = 12$ (implying the increase of 108 to 2556 coefficients). Analogously to Table 1, we observe a soft trend of over-fitting from $R = 12$ to $R = 14$ with $S = 3$, because more coefficients did not provide a better score over the test set.

In Figure 1, we present the average explained variance score over 100 Monte Carlo simulations, regarding all configurations, for the training and test sets. Note that the respective standard deviation is represented in this figure in order to stress the influence of the selection of sets and the initialization of the model coefficients. In accordance with Figure 1, as previously discussed, more coefficients considered in the TT network lead to an improvement in the performance of the training set; in contrast with that, the performance of the test set tends to saturate from $R = 14$ and $R = 12$ for $S = 2$ and $S = 3$ respectively. In other words, the use of more than 1400 and 2556 coefficients for $S = 2$ and $S = 3$ does not improve the test set prediction - hence, to use more coefficients is pointless.

It is interesting to observe the potential of contraction of the TT structures regarding a (10-200-1) NN with 2401 coefficients: it can be modeled as a TT network with much less coefficients. For $R = 2$ and $S = 3$, the TT network has only 108 coefficients, which represents less than 5% of the total number of neural network coefficients, and can achieve an average score for the test set equals to 0.8110 and 0.8958, regarding Tanh and ReLU functions. The best average performance for the test set is obtained for $S = 3$ and $R = 12$, with 2556 coefficients, with a score equal to 0.9126 and 0.9884 for, respectively, both tanh and ReLU functions.

Furthermore, Figure 1 also allows to better understand the influence of the parameter $S$, i.e. the dimension array of the encoded features. This parameter controls the degree level of the polynomial regression model, i.e. the level of non-linearity introduced by the feature mapping, and can enable to fit better the data interactions with lower number of coefficients, as shown in Figure 1.
Table 1: Recovering the (10-200-1) neural network with Tanh function

| R   | no. of training | no. of validation | no. of test | coeffs. | MSE | Score | SPCC |
|-----|----------------|-------------------|-------------|---------|-----|-------|------|
| 4   | 812            | 816               | 820         | 957     | 305 | 0.8418| 0.9176|
| 6   | 1237           | 1241              | 1245        | 1468    | 185 | 0.8682| 0.9394|
| 8   | 1672           | 1676              | 1680        | 1901    | 224 | 0.8922| 0.9480|
| 10  | 2107           | 2111              | 2115        | 2336    | 262 | 0.9149| 0.9559|
| 12  | 2542           | 2546              | 2550        | 2771    | 301 | 0.9370| 0.9629|
| 14  | 2977           | 2981              | 2985        | 3206    | 345 | 0.9589| 0.9699|

Table 2: Recovering the (10-200-1) neural network with ReLU function

| R   | no. of training | no. of validation | no. of test | coeffs. | MSE | Score | SPCC |
|-----|----------------|-------------------|-------------|---------|-----|-------|------|
| 4   | 812            | 816               | 820         | 957     | 305 | 0.8418| 0.9176|
| 6   | 1237           | 1241              | 1245        | 1468    | 185 | 0.8682| 0.9394|
| 8   | 1672           | 1676              | 1680        | 1901    | 224 | 0.8922| 0.9480|
| 10  | 2107           | 2111              | 2115        | 2336    | 262 | 0.9149| 0.9559|
| 12  | 2542           | 2546              | 2550        | 2771    | 301 | 0.9370| 0.9629|
| 14  | 2977           | 2981              | 2985        | 3206    | 345 | 0.9589| 0.9699|
5.2. Mackey-Glass noisy chaotic time series

In the second experiment, we consider the Mackey-Glass noisy chaotic time series in order to compare both neural and tensor networks, which refers to the following delayed differential equation Mackey and Glass (1977):

$$\frac{dx(t)}{dt} = x(t + \Delta t) = a \frac{x(t - \tau)}{1 + x(t - \tau)^n} - bx(t).$$  \hspace{1cm} (30)

The Mackey-Glass time series with 1000 samples was generated using the 4-th order Runge-Kutta method with the power factor $n=10$, initial condition $x(0)=1.2$, delay constant $\tau=17$, time step size $\Delta t=1.0$, and other parameters $a=0.2$ and $b=0.1$. According to Mirzaee (2009); Farmer (1982), for $\tau \geq 17$, the time series shows chaotic behavior. We consider four non-consecutive points of the time series, spaced by 6 points, with the aim of generating each input vector to predict the short-term $x(t+6)$ and long-term $x(t+84)$.
predictions, i.e.

\[ x(t + 6) = F(x(t - 18), x(t - 12), x(t - 6), x(t)) \]
\[ x(t + 84) = F(x(t - 18), x(t - 12), x(t - 6), x(t)) , \]

which represents a usual test \cite{Gu2007, Mirzaee2009, Ko2011}. The noiseless case is considered, as well as experiments with additive white Gaussian noise with zero mean and three values of standard deviation i.e. \( \sigma_N \in \{0.01, 0.05, 0.1\} \).

Three different 4-th order TT networks with \( (S=2,R=2) \), \( (S=2,R=4) \), \( (S=3,R=4) \) are employed to predict the short and long-term indices, as well as three different neural networks: (4-4-1), (4-6-1), (4-15-1) with two activation functions: \( \text{Tanh} \) and \( \text{ReLU} \). The choice of these neural network parameters is due to the restriction of one hidden layer, as discussed above, and the TT parameters come from the approximate number of coefficients, i.e. \( (24, 40, 90) \) and \( (25, 37, 91) \) for the TT and NN structures respectively.

Analogously to the previous subsection, the regularization factor search for the tensor network follows the same described procedure, regarding the validation set i.e. it is based on the GSS with a rough preliminary search from the same interval. We also adopted this procedure for the neural networks in order to search an optimal learning rate applied on the SGD method.

In Tables 3 and 4, we present all the results in terms of MSE, and \( \text{score} \), and SPCC at the convergence, for training, validation and test sets, for the short-term \( x(t+6) \) and long-term \( x(t+84) \) predictions respectively.

All results represent the average over 400 Monte Carlo simulations, which implies 400 different random initializations. As expected, the performance for all models are affected with the noise addition, specially with \( \sigma_N=0.1 \).

For the noiseless case, the best performance is achieved with the 4-th order TT \((S=3,R=4)\) with the \textit{score} 0.9972 and 0.8739 for short-term and long-term predictions of test sets respectively. For \( \sigma_N=0.1 \), the best \textit{score} is 0.6916 obtained with the (4-15-1) NN with ReLU for short-term predictions of test sets. However, for long-term predictions (and \( \sigma_N=0.1 \)), the 4-th order TT with \((S=3,R=4)\) provides the best \textit{score} with 0.6868. Consequently, the TT model tends to provide better performances than both NN models for the configuration with more coefficients, i.e. \((S=3,R=4)\).

The biggest \textit{score} difference between \( \sigma_N=0.0 \) and \( \sigma_N=0.1 \) is achieved regarding the 4-th order TT model with \((S=2,R=2)\) with 35.10\% for short-term predictions and the (4-15-1) NN model with \( \text{Tanh} \) with 26.42\% for long-term predictions of test sets, with respect to the noiseless case.

The short-term predictions tend to provide better results, as well as the increase of coefficients. From 24/25 to 90/91 coefficients, in the best scenario, we can increase the \textit{score} until 7.23\% and 6.35\% with the 4-th order TT model, 3.01\% and 1.18\% with the NN model with ReLU, and 0.38\% and 0.23\% with the NN model with \( \text{Tanh} \), for both short-term and long-term predictions of test sets respectively. Thus, the increment of coefficients for the TT models tends to provide a bigger improvement on the test sets compared to the NN models.

Figures 2-3 show the amplitude versus time for the Mackey-Glass time series at the convergence, for the training and test sets, regarding the short-term and long-term predictions, and with respect to the noiseless case \( (\sigma_N=0.0) \) and the noise standard deviation \( \sigma_N=0.1 \), respectively. The original targets (referred to in the figures as \textit{exact value}) were re-scaled into the range \([-1,1]\) and added a Gaussian noise (referred to as \textit{noisy target}) with respect to the standard deviation \( \sigma_N \). Note that each prediction curve represents the average over all Monte Carlo simulations with its respective standard deviation in order to emphasize the influence of initialization. The estimates, given by all models, tend to follow the oscillations in time of Mackey-Glass time series. The additional noise makes the forecast harder as well as the long-term predictions.

The convergence of Mackey-Glass series for all configurations is represented by Figures 4-5, regarding the short-term and long-term predictions, with respect to the noiseless case and the noise standard deviation \( \sigma_N=0.1 \). All the curves represent the average results, in terms of MSE and \textit{score} over all Monte Carlo simulations, the mean of MSE and \textit{score} at the convergence and its respective standard deviation are denoted in the legend.

According to these figures, TT structures are faster than NN models for all configurations. We can observe that less than 10 sweeps are enough to achieve the convergence for all TT structures and, in the
best case, only 2 sweeps. In contrast, NN networks with ReLU and \( \text{Tanh} \) respectively require at least 150 and 250 epochs in the best scenario. The ReLU function provides a better convergence than \( \text{Tanh} \), specially for short-term prediction. Furthermore, it is interesting to notice that the average performance is more representative for the TT model since the standard deviation is quite small, i.e. lower than four decimal places as indicated in the legend. Consequently, according to both figures, the initialization of coefficients in the neural networks tends to have more impact on the performance then in the tensor network, specially in the case of more coefficients and long-term predictions.

Figure 2: Short-term prediction \( x(t + 6) \) of Mackey-Glass time series (noiseless case) with 1000 samples (600 training + 200 validation + 200 test): average over 400 Monte Carlo simulations. Figs. (a) and (b) show the results comparing the TT model \( (S=2,R=2) \) and the \( (4-4-1) \) NN model for the training and test sets respectively. Figs. (c) and (d) show the results comparing the TT model \( (S=3,R=4) \) and the \( (4-15-1) \) NN model for the training and test sets respectively.
Figure 3: Long-term prediction $x(t + 84)$ of Mackey-Glass time series ($\sigma_N = 0.1$) with 1000 samples (600 training + 200 validation + 200 test): average over 400 Monte Carlo simulations. Figs. (a) and (b) show the results comparing the TT model ($S=2, R=2$) and the (4-4-1) NN model for the training and test sets respectively. Figs. (c) and (d) show the results comparing the TT model ($S=3, R=4$) and the (4-15-1) NN model for the training and test sets respectively.
| models               | σN | training               | validation               | test               |
|----------------------|----|------------------------|--------------------------|--------------------|
|                      |    | MSE | SCORE | SPCC | MSE | SCORE | SPCC | MSE | SCORE | SPCC |
| (4-4-1) NN           | 0.0| 1.387e-02 | 0.9712 | 0.9862 | 1.422e-02 | 0.9705 | 0.9859 | 1.412e-02 | 0.9701 | 0.9857 |
| with ReLU            | 0.05| 2.527e-02 | 0.8608 | 0.9388 | 2.595e-02 | 0.8778 | 0.9375 | 2.602e-02 | 0.8789 | 0.9371 |
| 25 coeffs.           | 0.1| 5.252e-02 | 0.6844 | 0.8282 | 5.430e-02 | 0.6737 | 0.8225 | 5.462e-02 | 0.6714 | 0.8213 |
| (4-4-1) NN           | 0.0| 3.893e-03 | 0.9882 | 0.9941 | 4.038e-03 | 0.9878 | 0.9939 | 3.985e-03 | 0.9878 | 0.9939 |
| with Tanh            | 0.05| 2.143e-02 | 0.8960 | 0.9466 | 2.204e-02 | 0.8933 | 0.9455 | 2.212e-02 | 0.8929 | 0.9453 |
| 25 coeffs.           | 0.1| 4.859e-02 | 0.0031 | 0.825 | 5.024e-02 | 0.0201 | 0.8294 | 5.609e-02 | 0.0241 | 0.8282 |
| (4-15-1) NN          | 0.0| 3.966e-03 | 0.9851 | 0.9926 | 3.990e-03 | 0.9851 | 0.9925 | 4.218e-03 | 0.9830 | 0.9915 |
| with ReLU            | 0.05| 2.327e-02 | 0.8286 | 0.9395 | 2.722e-02 | 0.8611 | 0.9290 | 2.598e-02 | 0.8592 | 0.9271 |
| 37 coeffs.           | 0.1| 5.174e-02 | 0.6743 | 0.8221 | 5.502e-02 | 0.6842 | 0.8186 | 4.509e-02 | 0.6630 | 0.7988 |
| (4-15-1) NN          | 0.0| 4.206e-03 | 0.9893 | 0.9946 | 4.328e-03 | 0.9889 | 0.9945 | 4.367e-03 | 0.9888 | 0.9944 |
| with Tanh            | 0.05| 2.101e-02 | 0.8974 | 0.9473 | 2.131e-02 | 0.8954 | 0.9465 | 2.162e-02 | 0.8945 | 0.9461 |
| 37 coeffs.           | 0.1| 4.877e-02 | 0.6963 | 0.8344 | 5.055e-02 | 0.6857 | 0.8290 | 5.072e-02 | 0.6851 | 0.8286 |
| (4-7-2) NN           | 0.0| 2.677e-03 | 0.9900 | 0.9950 | 2.719e-03 | 0.9988 | 0.9949 | 2.962e-03 | 0.9881 | 0.9940 |
| for (S=2, R=2)       | 0.05| 2.140e-02 | 0.8920 | 0.9445 | 2.347e-02 | 0.8802 | 0.9387 | 2.302e-02 | 0.8752 | 0.9356 |
| 40 coeffs.           | 0.1| 4.939e-02 | 0.0890 | 0.8301 | 5.209e-02 | 0.6651 | 0.8169 | 5.138e-02 | 0.6549 | 0.8096 |
| (4-15-1) NN          | 0.0| 3.306e-03 | 0.9918 | 0.9960 | 3.460e-03 | 0.9913 | 0.9958 | 3.437e-03 | 0.9913 | 0.9958 |
| with ReLU            | 0.05| 1.915e-02 | 0.9339 | 0.9598 | 2.027e-02 | 0.8997 | 0.9483 | 2.035e-02 | 0.8975 | 0.9477 |
| 91 coeffs.           | 0.1| 4.565e-02 | 0.7098 | 0.8425 | 4.818e-02 | 0.6913 | 0.8336 | 4.856e-02 | 0.6916 | 0.8326 |
| (4-15-1) NN          | 0.0| 3.880e-03 | 0.9904 | 0.9952 | 3.959e-03 | 0.9899 | 0.9950 | 3.945e-03 | 0.9900 | 0.9951 |
| with Tanh            | 0.05| 2.118e-02 | 0.8997 | 0.9486 | 2.175e-02 | 0.8976 | 0.9478 | 2.174e-02 | 0.8963 | 0.9470 |
| 91 coeffs.           | 0.1| 5.030e-02 | 0.8093 | 0.8302 | 5.178e-02 | 0.6820 | 0.8267 | 5.106e-02 | 0.6795 | 0.8255 |
| (4-TT)               | 0.0| 5.966e-04 | 0.9778 | 0.9989 | 5.857e-04 | 0.9778 | 0.9989 | 6.985e-04 | 0.9972 | 0.9986 |
| for (S=3, R=4)       | 0.05| 1.834e-02 | 0.9074 | 0.9526 | 1.888e-02 | 0.9036 | 0.9506 | 1.977e-02 | 0.8927 | 0.9453 |
| 90 coeffs.           | 0.1| 4.567e-02 | 0.7125 | 0.8442 | 4.568e-02 | 0.7060 | 0.8404 | 4.708e-02 | 0.6842 | 0.8274 |
### Table 4: Mackey-Glass time series for long-term prediction

| models       | $\sigma_N$ | training | validation | test   |
|--------------|------------|----------|------------|--------|
|              |            | MSE      | SCORE      | SPCC   | MSE      | SCORE      | SPCC   |
| (4-4-1) NN   | 0.0        | 4.320e-02| 0.8462     | 0.9299 | 4.561e-02| 0.8382     | 0.9170 |
|              |            |          |            |        | 4.598e-02| 0.8390     | 0.9174 |
| with ReLU    | 0.05       | 4.502e-02| 0.7789     | 0.8832 | 4.644e-02| 0.7718     | 0.8809 |
|              |            |          |            |        | 4.610e-02| 0.7736     | 0.8810 |
| 25 coeffs.   | 0.1        | 6.022e-02| 0.6283     | 0.7934 | 6.192e-02| 0.6231     | 0.7914 |
|              |            |          |            |        | 6.235e-02| 0.6210     | 0.7899 |
| (4-4-1) NN   | 0.0        | 4.099e-02| 0.8468     | 0.9283 | 4.039e-02| 0.8382     | 0.9170 |
| with Tanh    | 0.05       | 4.280e-02| 0.7796     | 0.8830 | 4.376e-02| 0.7770     | 0.8822 |
|              |            |          |            |        | 4.376e-02| 0.7773     | 0.8813 |
| 25 coeffs.   | 0.1        | 5.823e-02| 0.6286     | 0.7935 | 5.936e-02| 0.6222     | 0.7902 |
|              |            |          |            |        | 5.942e-02| 0.6211     | 0.7901 |
| 4-TT         | (S=2, R=2) | 0.0      | 3.556e-02  | 0.8679 | 3.930e-02| 0.8483     | 0.9211 |
|              |            |          |            |        | 4.091e-02| 0.8218     | 0.9065 |
|              |            | 0.05     | 4.264e-02  | 0.7806 | 4.207e-02| 0.7803     | 0.8834 |
|              |            |          |            |        | 4.125e-02| 0.7937     | 0.8929 |
|              |            | 0.1      | 5.950e-02  | 0.6187 | 5.513e-02| 0.6419     | 0.8014 |
|              |            |          |            |        | 5.255e-02| 0.6755     | 0.8287 |
| (4-6-1) NN   | 0.0        | 3.982e-02| 0.8506     | 0.9223 | 4.174e-02| 0.8443     | 0.9194 |
| with ReLU    | 0.05       | 4.342e-02| 0.7822     | 0.8848 | 4.536e-02| 0.7735     | 0.8805 |
|              |            |          |            |        | 4.500e-02| 0.7749     | 0.8814 |
| 37 coeffs.   | 0.1        | 5.815e-02| 0.6354     | 0.7973 | 6.055e-02| 0.6225     | 0.7905 |
|              |            |          |            |        | 6.033e-02| 0.6216     | 0.7899 |
| (4-6-1) NN   | 0.0        | 3.9734e-02| 0.8482  | 0.9211 | 4.110e-02| 0.8425     | 0.9185 |
| with Tanh    | 0.05       | 4.207e-02| 0.7800     | 0.8832 | 4.334e-02| 0.7760     | 0.8816 |
|              |            |          |            |        | 4.357e-02| 0.7768     | 0.8822 |
| 40 coeffs.   | 0.1        | 5.817e-02| 0.6283     | 0.7929 | 5.870e-02| 0.6274     | 0.7832 |
|              |            |          |            |        | 5.945e-02| 0.6242     | 0.7914 |
| (4-15-1) NN  | 0.0        | 3.782e-02| 0.8566     | 0.9255 | 3.987e-02| 0.8483     | 0.9215 |
| with ReLU    | 0.05       | 4.178e-02| 0.7871     | 0.8872 | 4.389e-02| 0.7763     | 0.8818 |
|              |            |          |            |        | 4.416e-02| 0.7765     | 0.8819 |
| 91 coeffs.   | 0.1        | 5.642e-02| 0.6400     | 0.8001 | 5.860e-02| 0.6291     | 0.7943 |
|              |            |          |            |        | 5.916e-02| 0.6269     | 0.7931 |
| (4-15-1) NN  | 0.0        | 3.964e-02| 0.8485     | 0.9212 | 4.064e-02| 0.8451     | 0.9197 |
| with Tanh    | 0.05       | 4.287e-02| 0.7800     | 0.8832 | 4.389e-02| 0.7751     | 0.8810 |
|              |            |          |            |        | 4.355e-02| 0.7770     | 0.8823 |
| 91 coeffs.   | 0.1        | 5.821e-02| 0.6287     | 0.7931 | 5.928e-02| 0.6230     | 0.7905 |
|              |            |          |            |        | 5.942e-02| 0.6211     | 0.7895 |
| 4-TT         | (S=3, R=4) | 0.0      | 2.471e-02  | 0.9055 | 2.797e-02| 0.8920     | 0.9445 |
|              |            |          |            |        | 3.320e-02| 0.8739     | 0.9355 |
|              |            | 0.05     | 3.827e-02  | 0.8011 | 3.731e-02| 0.8064     | 0.8980 |
|              |            |          |            |        | 3.730e-02| 0.8136     | 0.9040 |
|              |            | 0.1      | 5.706e-02  | 0.6343 | 5.367e-02| 0.6532     | 0.8087 |
|              |            |          |            |        | 5.072e-02| 0.6868     | 0.8374 |
Figure 4: Convergence analysis of Mackey-Glass time series (noiseless case), with 1000 samples (600 training + 200 validation + 200 test), for short-term prediction: average over 400 Monte Carlo simulations. Figs. (a),(b) and (c),(d) show the results comparing two TT models and four NN models for the training and test sets respectively.

5.3. NASDAQ index forecasting

In the last decade, neural networks have been applied to the forecast of financial time series because of the ability to perform complex nonlinear modeling and to capture the underlying pattern within a time series, providing a powerful statistical modeling technique that can be an alternative to traditional methodologies, such as ARMA, ARCH and Generalized ARCH (GARCH) models Tang et al. (1991); Gately (1995); Chatterjee et al. (2000).

The goal of this section is to analyze the performance of a TT network, in a real case, in forecasting financial time series; moreover, it is interesting to compare its performance with that of the neural network model. The data were obtained from finance.yahoo.com. The input variables of networks are given by four past values of the time series, spaced in $\Delta$ samples, which are selected through auto-correlation analysis in terms of sample Pearson correlation.

We have considered a period of a daily closing stock market of NASDAQ in USD, for short and long-term predictions, from January 2, 2018 until December 28, 2018 with $\Delta=1$ for daily predictions $x(t+1)$ and with...
Figure 5: Convergence analysis of Mackey-Glass time series (\(\sigma_N=0.1\)), with 1000 samples (600 training + 200 validation + 200 test), for long-term prediction: average over 400 Monte Carlo simulations. Figs. {(a),(b)}, and {(c),(d)} show the results comparing two TT models and four NN models for the training and test sets respectively.

\[ \Delta=30 \] for monthly predictions \(x(t+30)\), i.e.

\[
\begin{align*}
x(t+1) &= F(x(t-3), x(t-2), x(t-1), x(t)) \\
x(t+30) &= F(x(t-90), x(t-60), x(t-30), x(t)).
\end{align*}
\]

The training, validation and test sets were randomly selected from the input data and we have applied 200 Monte Carlo simulations, implying 200 different random sets with different initializations for weighting coefficients, in order to mitigate the influence of weighting initialization and the chosen sets on the algorithms.

We apply the same procedure for selecting an optimal regularization factor, associated to the TT model, based on the searching step, described earlier, regarding the same input interval \(\{2^n : n \in \mathbb{Z}, -10 \leq n \leq 10\}\) and considering the lowest MSE obtained from the validation set. Unlike the previous subsection, this problem requires a faster algorithm for learning NNs, with adaptive update of the learning rate; hence, we employed the Adam algorithm (originally proposed in Kingma and Ba (2014)) given in Reddi et al. (2018), which is a modified version without the debiased step of the original version, with the following hyper-parameters, typically recommended in practice Kingma and Ba (2014); Reddi et al. (2018): the initial learning rate
Five different structures have been chosen for the TT and NN models and employed to predict the short and long-term indices, with approximate number of coefficients, i.e. $(24, 90, 544, 1300)$ and $(25, 91, 181, 547, 1303)$ for both respective structures. For the TT model, we have: $(S=2, R=2), (S=3, R=4), (S=4, R=16), (S=5, R=25)$. For the NN model, we have: $(4-4-1), (4-15-1), (4-30-1), (4-91-1), (4-217-1)$ with two activation functions $\text{Tanh}$ and ReLU.

In Tables 5 and 6, all results are shown in terms of MSE, score, SPCC, and $R$-squared at the convergence, for training, validation, and test sets, for the short-term $x(t + 1)$ and long-term $x(t + 30)$ predictions respectively. According to Table 5, we can note that the performance of both models for the daily prediction does not have a significant improvement on the training set with the increase of coefficients, from $25/24$ to $1303/1300$, mainly for the TT and NN model with $\text{Tanh}$ function, lower than two decimal places.

Furthermore, it is possible to check a decrement on the performance of training and test sets when more coefficients are employed, regarding the average score respectively of the validation and test sets, of 1.67% and 1.81% for the NN with ReLU, 0.11% and 0.16% for the NN with $\text{Tanh}$, and 0.78% and 1.72% for the TT model. These decays indicate a tendency to over-fitting of all models: thus, more coefficients will not provide better results associated to the test set. The best performance regarding the test sets is obtained with the $(4-4-1)$ NN model with $\text{Tanh}$ with the score 0.9243, followed by the $(4-4-1)$ NN with ReLU with 0.9212 and the 4-th order TT model with $(S=2, R=2)$ with 0.9200, respectively representing a reduction of 0.34% and 0.46% with respect to the best score.

In contrast, taking into account Table 6, we verify a simultaneous improvement for the monthly predictions on the training, validation, and test sets, except to the NN model with ReLU. For this last structure, we observe a decay of the performance on the validation and test sets when we employ more than 30 hidden neurons in the intermediate layer. Therefore, the best result is achieved with the highest number of coefficients only with the NN model with $\text{Tanh}$ and the TT model. The $(4-217-1)$ NN model with $\text{Tanh}$, the 4-th order TT model with $(S=5, R=25)$, and the $(4-30-1)$ NN model with ReLU respectively provide a score 0.8465, 0.8458 and 0.8501, which represent an increment on the test set of 5.54%, 7.29%, 8.21% respectively with respect to the worst configuration, i.e. the case with the lowest number of coefficients for each model. Note that this improvement was achieved by the increase of coefficients, from $24/25$ to $1303/181/1300$ coefficients for respectively the TT, the NN with ReLU and the NN with $\text{Tanh}$ models. Therefore, both TT and NN with $\text{Tanh}$ provide similar performances, but the TT showed a higher increment on the performance of test sets when more coefficients are considered.

Figures 6 and 7 represent the relation between the short-term $\hat{x}(t+1)$ and long-term $\hat{x}(t+30)$ predictions, with the respective standard deviations, versus the desired target $(x(t+1)$ or $x(t+30))$ by separately taking into account the predictions of the training, validation, test and all sets for each model. The best-fitting (or regression) line and the fitted line associated to each prediction, through the slope $m$ and the $y$-intercept $b$ of each red line, are indicated in each chart. Note that only the best configuration for each model is presented in this figure, for daily and monthly predictions, as discussed above. It is important to emphasize that this kind of chart presents a visualization resource for the learned predictions and it will not necessarily point out the same best model since the best-fitting line is given by a straight line, which linearly maps the error of predictions.

When the prediction is closer enough to the desired value, the slope tends to one as well as the $y$-intercept tends to zero, thus, in the ideal case, we have $m \approx 1$ and $b \approx 0$. From these figures, we verify, as expected, that the predictions of the training set (even as all data sets) provide better fitting performance once both ideal and fitted lines are closer than the lines associated to the predictions of the validation and test sets. Furthermore, as also expected, we obtain worse performances for monthly predictions than the daily predictions.
### Table 5: NASDAQ index for short-term prediction

| models | no. of coeffs. | training | validation | test |
|--------|----------------|----------|------------|------|
| (4-4-1) NN ReLU | 25 | 8.948e-03 | 0.9395 | 0.9693 | 0.9130 | 0.9222 | 0.9627 | 0.9200 | 1.134e-02 | 0.9212 | 0.9616 | 0.9191 |
| (4-15-1) NN ReLU | 91 | 7.931e-03 | 0.9464 | 0.9728 | 0.9286 | 1.203e-02 | 0.9174 | 0.9605 | 0.9150 | 1.246e-02 | 0.9147 | 0.9591 | 0.9124 |
| (4-30-1) NN ReLU | 181 | 7.394e-03 | 0.9500 | 0.9747 | 0.9200 | 1.220e-02 | 0.9163 | 0.9603 | 0.9138 | 1.286e-02 | 0.9132 | 0.9582 | 0.9108 |
| (4-91-1) NN ReLU | 547 | 6.662e-03 | 0.9550 | 0.9772 | 0.9249 | 1.258e-02 | 0.9118 | 0.9583 | 0.9093 | 1.328e-02 | 0.9086 | 0.9562 | 0.9062 |
| (4-217-1) NN ReLU | 1303 | 5.956e-03 | 0.9597 | 0.9796 | 0.9287 | 1.361e-02 | 0.9068 | 0.9561 | 0.9040 | 1.438e-02 | 0.9045 | 0.9543 | 0.9020 |

### Table 6: NASDAQ index for long-term prediction

| models | no. of coeffs. | training | validation | test |
|--------|----------------|----------|------------|------|
| (4-4-1) NN ReLU | 25 | 3.335e-02 | 0.8360 | 0.9115 | 0.8386 | 4.316e-02 | 0.7911 | 0.8976 | 0.7780 | 4.489e-02 | 0.7956 | 0.8933 | 0.7709 |
| (4-15-1) NN ReLU | 91 | 1.715e-02 | 0.8713 | 0.8967 | 0.8926 | 2.691e-02 | 0.8417 | 0.9284 | 0.8315 | 3.214e-02 | 0.8469 | 0.9313 | 0.8386 |
| (4-30-1) NN ReLU | 181 | 1.358e-02 | 0.8736 | 0.8962 | 0.8936 | 3.169e-02 | 0.8563 | 0.9312 | 0.8401 | 3.728e-02 | 0.8591 | 0.9318 | 0.8421 |
| (4-91-1) NN ReLU | 547 | 9.251e-03 | 0.9148 | 0.9072 | 0.9048 | 3.315e-02 | 0.8817 | 0.9315 | 0.8315 | 3.934e-02 | 0.8869 | 0.9313 | 0.8386 |
| (4-217-1) NN ReLU | 1303 | 6.791e-03 | 0.9469 | 0.9137 | 0.9101 | 3.423e-02 | 0.8530 | 0.9260 | 0.8248 | 3.976e-02 | 0.8583 | 0.9271 | 0.8302 |

24
Figure 6: Short-term prediction $x(t+1)$ of NASDAQ index with 246 samples (148 training + 49 validation + 49 test): average over 200 Monte Carlo simulations. Figs. {(a)-(d)}, {(e)-(h)} and {(i)-(l)} show respectively the results for the (4-4-1) NN model with ReLU, the (4-4-1) NN model with Tanh, and the TT model ($S=2, R=2$).
Figure 7: Long-term prediction $x(t + 30)$ of NASDAQ index with 130 samples (78 training + 26 validation + 26 test): average over 200 Monte Carlo simulations. Figs. {(a)-(d)}, {(e)-(h)} and {(i)-(l)} show respectively the results for the (4-30-1) NN model with ReLU, the (4-217-1) NN model with Tanh, and the TT model ($S=5$, $R=25$).
Figure 6 indicates the best fit of slope and y-intercept achieved for the test set with the NN model with ReLU \((m = 0.88, b = 909.33)\), followed by the NN with Tanh with \((m = 0.86, b = 1013.26)\) and the TT model with \((m = 0.82, b = 1290.47)\). According to Figure 7, by considering long-term predictions, the best learned slope and y-intercept for the test set is achieved with the NN model with ReLU with \((m = 0.43, b = 3922.86)\), followed by the NN with Tanh with \((m = 0.37, b = 4330.67)\) and the TT model with \((m = 0.35, b = 4520.26)\).

The NASDAQ index predictions of all data set over the selected time period at the convergence for both short- and long-term predictions are presented in Figures 8-9, for only two different configurations for each structure. The original target is also represented in these figures as well as the average MSE, score and \(R^2\) of test sets over all Monte Carlo simulations. Comparing both figures, observe that the standard deviation of the predictions are more visible for monthly predictions, i.e. \(x(t+30)\), than for daily predictions \(x(t+1)\). Despite the difference of performance between all models, we can observe that the learned models follow the oscillations of the index in time, mainly for daily forecast. Furthermore, in Figure 9, we can note a visual difference between both predictions with 24/25/25 and 1303/181/1300 coefficients for respectively the TT/NN with ReLU and NN with Tanh models, unlike the short-term predictions.

Figures 10 and 11 show the convergence of NASDAQ index forecasting of short-term and long-term for the training and test sets. The averages of MSE and score over all Monte Carlo simulation are shown in these figures and we denote the mean of MSE and score at the convergence and its respective standard deviation in the legend. Clearly, the TT models present the fastest convergence, the maximum of 6 sweeps is required; on the other hand, the NN models with ReLU and Tanh require more 2000 epochs in the worst scenario. It is interesting to observe that the standard deviations, denoted in the figures, shown the proximity of the results at the convergence and the influence of random initialization of weighting networks and the selection of the datasets.

6. Conclusions

A key feature of this paper is the development of a modeling approach for NNs through its input/output with the aim of promoting the recoverability of NNs with lower complexity of network structure. Another aspect addressed the machine learning models based on the tensor approach with the purpose of learning a non-linear predictive model directly from a given data, by restricting it to a given tensor structure, in our case, TT network.
Expressions were derived in order to view the non-linear global problem for estimating tensor structure as a set of several sub-problems for estimating each core by means of a closed-form solution based on normal equation, analogously to a standard regression model. It can allow the extension of applicability of TT structures to machine learning algorithms. Furthermore, these expressions provide a reduction of required memory and computational complexity. The addition of a matrix regularization factor in the loss function enables a parameter to adjust the model flexibility for the purpose of providing a balance between training performance and capability of the model generalization, i.e. by preventing over-fitting problem.

From the first part of our simulations, MLPs were modeled by TT networks, which enabled a compact representation of a simple MLP. The smallest adopted TT network with the lowest number of coefficients, representing a reduction of 95% of NN coefficients, provided an average score over the test set equal to 0.8110 and 0.8958 depending on the adopted activation function (i.e. Tanh and ReLU functions respectively). The best score, achieved with 20% of reduction in terms of number coefficients, is 0.9101 and 0.9880 for both Tanh and ReLU functions respectively. Furthermore, we verified the relevance of non-linearity introduced by feature mapping, which can enable a better model fitting with lower number of coefficients.

The second part was driven by applications in noisy chaotic time series and stock market index price forecasting, by means of Mackey-Glass equation and NASDAQ index. The estimates, given by neural and tensor networks, tend to follow the oscillations in time of Mackey-Glass time series. As expected, the additional noise makes the forecast harder as well as the long-term predictions. TT structures tended to provide better performances over test sets regarding networks with more coefficients. In addition, the increment of coefficients for the TT models tends to provide a bigger improvement on the test sets compared to the NN models. Besides that, for Mackey-Glass time series, we have observed that the initialization of coefficients in the neural networks tends to have more impact on the performance then in the tensor network, specially in the case of more coefficients and long-term predictions.

From the results regarding the NASDAQ index forecasting, we can note that the performance of both NN and TT models for the daily prediction does not have a significant improvement on the training set with the increase of coefficients, compared to the monthly prediction. Differently to the previous time series, we have noted a tendency to over-fitting of all models, more coefficients did not provide better results associated to the test set. The best performance over test sets, regarding all networks, for daily predictions is achieved with the lowest number of coefficients. In contrast, we verify a simultaneous improvement for the monthly predictions on the training, validation, and test sets, except to the NN model with ReLU. For this last
structure, we observe a decay of the performance on the validation and test sets when we employ more than 30 hidden neurons in the intermediate layer. Therefore, the best result is achieved with the highest number of coefficients only with the NN model with Tanh and the TT model. Besides almost the same performance regarding the last models, the TT showed a higher increment on the performance over test sets when more coefficients are considered.

In terms of convergence speed, tensor networks tend to achieve faster convergence, thanks to the closed-form solution. We also observed that neural networks are quite sensitive to the adjustment of hyper-parameters and the importance of choice of adaptive learning-method algorithms in order to accelerate the convergence by adapting the learning rate. When we consider more sophisticated methods, the algorithms tend to be more robust, on the other hand, more hyper-parameters will be probably required.

By concluding, tensor networks are promising to design architectures of the DNNs more efficiently, and also they can accelerate and simplify the learning process in terms of network structure, algorithm tuning procedure, computational complexity and required memory, while maintaining a reasonable quality of prediction.

Acknowledgements

This work was supported by FAPESP, Brazil [grant number 2014/23936-4] and by CNPq, Brazil [grant number 308811/2019-4].

Figure 10: Convergence analysis of NASDAQ index, with 246 samples (148 training + 49 validation + 49 test), for short-term prediction: average over 200 Monte Carlo simulations. Figs. {(a),(b)} and {(c),(d)} show the results comparing two TT models and four NN models for the training and test sets respectively.
Figure 11: Convergence analysis of NASDAQ index, with 130 samples (78 training + 26 validation + 26 test), for long-term prediction: average over 200 Monte Carlo simulations. Figs. {(a),(b)} and {(c),(d)} show the results comparing two TT models and four NN models for the training and test sets respectively.

References

Adya, M., Collopy, F.L., 1998. How effective are neural networks at forecasting and prediction? a review and evaluation. Journal of Forecasting 17, 481–495.

Aizerman, M.A., Braverman, E.A., Rozonoer, L., 1964. Theoretical foundations of the potential function method in pattern recognition learning, in: Automation and Remote Control, pp. 821–837.

Bishop, C.M., 2006. Pattern recognition and machine learning. Springer.

Björck, Å., 1988. A bidiagonalization algorithm for solving large and sparse ill-posed systems of linear equations. BIT Numerical Mathematics 28, 659–670. doi:10.1007/BF01941141.

Box, G., Jenkins, G., Reinsel, G., Ljung, G., 2015. Time Series Analysis: Forecasting and Control. Wiley Series in Probability and Statistics, Wiley. URL: https://books.google.com.br/books?id=rNt5CgAAQBAJ.

Cao, L.J., Tay, F.E.H., 2003. Support vector machine with adaptive parameters in financial time series forecasting. IEEE Transactions on Neural Networks 14, 1506–1518. doi:10.1109/TNN.2003.820556.

Chatterjee, A., Ayadi, O., E.Boone, B., 2000. Artificial neural network and the financial markets: A survey. Managerial Finance 26, 32–45. doi:10.1108/03074350010767034.

Chng, E.S., Chen, S., Mulgrew, B., 1996. Gradient radial basis function networks for nonlinear and nonstationary time series prediction. IEEE Transactions on Neural Networks 7, 190–194. doi:10.1109/72.478403.

Cichocki, A., 2014. Era of Big Data Processing: A new approach via tensor networks and tensor decompositions. Computing Research Repository (CoRR) URL: http://arxiv.org/abs/1403.2048.

Cortes, C., Vapnik, V., 1995. Support-vector networks. Machine Learning 20, 273–297. URL: https://doi.org/10.1007/BF00994018.

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. math cont sig syst (mcss) 2:303-314. Mathematics of Control, Signals, and Systems 2, 303–314. doi:10.1007/BF02551274.
Oseledets, I.V., Tyryshnikov, E., 2009. Breaking the curse of dimensionality, or how to use SVD in many dimensions. SIAM Journal on Scientific Computing 31, 3744–3759.

Pankratz, A., 1983. Forecasting with Univariate Box-Jenkins Models: Concepts and Cases. John Wiley, New York.

Polson, N.G., Scott, S.L., Rejoinder: Bayesian Anal., 43–47 doi:10.1214/11-BEAS01REJ.

Qian, N., 1999. On the momentum term in gradient descent learning algorithms. Neural Networks 12, 145 – 151. doi:https://doi.org/10.1016/S0893-6080(98)00116-6.

Raicharetn, T., Lursinsap, C., Sanguanbhokai, P., 2003. Application of critical support vector machine to time series prediction, in: Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS ’03., pp. V–V. doi:10.1109/ISCAS.2003.1206419.

Reddi, S., Kale, S., Kumar, S., 2018. On the convergence of adam and beyond, in: International Conference on Learning Representations.

Robinson, P., 1977. The estimation of a nonlinear moving average model. Stochastic Processes and their Applications 5, 81 – 90. URL: http://www.sciencedirect.com/science/article/pii/0304414977900527, doi:https://doi.org/10.1016/0304-4149(77)90052-7.

Savostyanov, D., Oseledets, I., 2011. Fast adaptive interpolation of multi-dimensional arrays in tensor train format, in: Multidimensional (nD) Systems (nDe), 2011 7th International Workshop on, IEEE, pp. 1–8.

Stoudenmire, E.M., Schwab, D.J., 2016. Supervised learning with quantum-inspired tensor networks. CoRR abs/1605.05775. URL: http://arxiv.org/abs/1605.05775, arXiv:1605.05775.

Suykens, J., Vandewalle, J., 1999. Least squares support vector machine classifiers. Neural Processing Letters 9, 293–300. URL: https://doi.org/10.1023/A:1018628609742, doi:10.1023/A:1018628609742.

Tang, Z., de Almeida, C., Fishwick, P., 1991. Time series forecasting using neural networks vs. box jenkins methodology. Transactions of The Society for Modeling and Simulation International - SIMULATION 57, 303–310. doi:10.1177/003754979105700508.

Tibshirani, R., 1996. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society, Series B 58, 267–288.

Tieleman, T., Hinton, G., 2012. Lecture 6.5 – RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Tsay, R., 2010. Analysis of Financial Time Series. CourseSmart, Wiley. URL: https://books.google.com.br/books?id=OKUGAwAAQBAJ.

Van Gestel, T., Suykens, J.A.K., Baestaens, D., Lambrechts, A., Lanckriet, G., Vandaele, B., De Moor, B., Vandewalle, J., 2001. Financial time series prediction using least squares support vector machines within the evidence framework. IEEE Transactions on Neural Networks 12, 809–821. doi:10.1109/72.935093.

Yang, Y., Hospedales, T., 2016. Deep multi-task representation learning: A tensor factorisation approach. Computing Research Repository (CoRR). URL: http://arxiv.org/abs/1605.06391.

Zhang, G.P., 2012. Neural networks for time-series forecasting, in: Handbook of Natural Computing. Springer, pp. 461–477.

Zhang, P., Patuwo, E., Hu, M., 1998. Forecasting with artificial neural networks: The state of the art. International Journal of Forecasting 14, 35–62. doi:10.1016/S0169-2070(97)00044-7.

Zou, H., Hastie, T., 2005. Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society, Series B 67, 301–320.