Evaluating Brain-Inspired Machine Learning Models for Time Series Forecasting: A Comparative Study on Dynamical Memory Within Reservoir Computing and Neural Networks

Eddie Nevander Hellström and Johan Slettengren

***Abstract*—Brain-inspired computing is a promising research field, with potential to encourage breakthroughs within machine learning and enable us to solve complex problems in a more efficient way. This study aims to compare the performance of brain-like machine learning algorithms for time series forecasting. Three models were implemented: a vanilla recurrent neural network (VRNN), a more brain-like reservoir computing (RC) model, as well as a time lagged version of the latter (TLRC). Additionally, an autoregressive integrated moving average (ARIMA) was implemented to obtain benchmark results, since this is a well established model with no connection to the brain. The performances were evaluated on a spectrum of univariate and multivariate time series, ranging from chaotic benchmark data to experimental data, such as temperature recordings. The results indicate that the reservoir computing models generally outperform the recurrent neural network, and that the considered models have a disadvantage in practical scenarios. A common factor for these algorithms is that they exhibit a sense of dynamical memory. Hence, we compare these models not only in terms of their predictive accuracy, but also in terms of their capability for memory. To reach an analysis of dynamical memory, this study investigated the usefulness of the algorithms on varying amounts of available information. Lastly, the effect of the network/reservoir size was taken into account. The research suggests that the models cannot benefit meaningfully from having more neurons.**

***Sammanfattning*—Beräkningsalgoritmer som är inspirerade av hjärnan har stor potential att bidra till genombrott inom maskininlärning och möjliggöra effektivare lösningar till komplexa problem. Denna studie syftar till att jämföra prestandan hos beräkningaralgoritmer som liknar hjärnan, tillämpade för tidsserieprediktion. Tre algortimer implementerades: ett standardimplementerat återkommande neuralt nätverk (VRNN), en reservoir computing (RC) model som efterliknar hjärnan och en tidsfördröjd version av den senare (TLRC). Utöver dessa användes även en autoregressive integrated moving average (ARIMA) modell, för att generera referensresultat, då modellen är väletablerad och saknar anknytning till hjärnan. Prestandan evaluerades på ett spektrum av envariabla- och multivariabla tidsserier, från kaotisk referensdata till experimentell data, såsom temperaturmätningar. Resultaten indikerar att reservoir computing modellerna generellt sett överträffar det återkommande neurala nätverket, samt att praktiska scenarier är ogynnsamma för de betraktade modellerna. En gemensam nämnare för dessa algortimer är att de besitter en typ av dynamiskt minne. Således jämfördes algoritmer inte enbart med hänsyn till deras prediktiva precision, men även i termer av minneskapacitet. För att nå en analys av dynamiskt minne undersökte denna studie de respektive modellernas kapacitet för varierande mängder av tillgänglig information. Till sist beaktades inverkan av nätverkets/reservoarens storlek. Resultaten tyder på att modellernas prestanda inte meningsfullt kunde dra nytta av ett större antal neuroner.**

***Index Terms*—brain-inspired computing, recurrent neural network, machine learning, reservoir computing, time series prediction, dynamical memory, autocorrelation**

Supervisors: Pawel Herman

TRITA number:

I. Introduction

The human brain has evolved to efficiently process information in order to make adequate decisions in complex and often ambiguous situations [1] [2]. The biological structure of the brain is thus a natural source of inspiration for machine learning algorithms, which has contributed to mayor breakthroughs within the field [2] [3]. Indeed, brain-inspired architectures constitute the foremost approach within many domains of artificial intelligence [4]. Owing to its historical success and current popularity, brain inspired computing is a promising research field, with the potential to encourage further breakthroughs and help us solve complex problems in a more efficient way. In the brain, information is processed by fundamental units of computation called neurons, which are connected to each other via synapses. The strength of their connections can be modified, a mechanism referred to as synaptic plasticity [5], which is the basis for learning [6]. When two connected neurons are simultaneously activated, there occurs a long term increase of effectiveness in their communication.

A time series dataset contains attribute values corresponding to different instances in time, within some dynamical system, such as business, economics or environment. One reason for analysing such series is to uncover patterns within the data, letting us accurately forecast behaviours of the system and enabling us to make informed decision about the future [7] [8]. Such prediction can be difficult however, especially when dynamics are of chaotic nature, as this means that the system is extremely sensitive to small perturbations, such as noise and different types of error. Pattern-recognition is crucial for time series forecasting and one significant patterns is *seasonality*, described as: ”a variation which repeats itself in systematic intervals over time” [9]. At the presence of seasonality, an *autocorrelation function* can be used to characterise the periodicity. The autocorrelation of a time series indicates how

strongly the data is correlated with parts of its own history or future [10]. The technicalities of this tool is discussed in more detail in the *Method* section, but the core idea is that the function has spikes at x-values corresponding to multiples of periods. Of course, a time series dataset – and especially experimental data – might correlate strongly with some part of its history, without satisfying the above definition of *seasonality*, e.g. by having seasons for only a limited juncture of time. In view of this, we will consider only the *time scale correlation* of a time series, given by its autocorrelation function. One might view this as a proxy for seasonality, in the case of irregular time series. Furthermore, a time-series may be *univariate*, meaning that it has one attribute per time step or *multivariate*, meaning that this number is greater than one. As stated in [11]: "Widespread interest in discovering features and trends in time series has generated a need for tools that support interactive exploration". In light of this, the purpose of our study is to investigate the effectiveness of different methods for prediction of time series data. More specifically two models are compared, namely a *Vanilla Recurrent Neural Network* (VRNN) and a *Reservoir Computing* (RC) model, which are discussed further in their respective sections.

Related literature includes [12], [13], [14] and [15]. A review of *recurrent neural networks* (RNNs) is provided by [12], including theoretical background as well as recent advances within the field, in terms of development of the model and practical results. A comprehensive introduction to reservoir computing, as well as practical recommendations for successful application of the model is provided by [13]. Similarly, a survey of techniques for generating and training RC models, as well as a discussion of important concepts surrounding this is carried out by [14]. Finally, found in [15] is a comparative study, investigating the use of different RNN- and RC models for chaotic time series prediction

Although the results produced in [15] are highly relevant to our investigation, we have identified three major limitations, which we aim to improve upon:

1. The study lacks references to benchmark results. A well-established standard is needed to put the study in a broader context.
2. Only one experimental dataset was used and corresponding results were inconclusive – "better performance was only obtained [...] at the cost of more computational time and effort". This results in a lack of practical relevance.
3. The notion of dynamical memory, is an important part of what makes the considered models effective when it comes to temporal forecasting. In [15] this property is discussed only theoretically, but not put in relation to practical results.

Listed below are the corresponding proposed improvements:

1. *Autoregressive integrated moving average* is a well established algorithm with no connection to the brain. We use it in this study to produce benchmark results.
2. Three different experimental dataset are considered in this study. We do not require these datasets to have a chaotic quality, so they can be chosen more freely, such that they give conclusive results.
3. In order to get a handle on the dynamical-memory-property of the models, we vary the amount of information used for forecasting. The idea here is that a model with a good capability for memory might be able to benefit from a large window of information. Furthermore, we will implement a *time lagged reservoir computing* model, aiming to utilise historical information in a more direct way than regular RC. Dynamical memory relates to the time scale correlations of time series, since a large window might be needed to accurately forecast a time series with long periods, favouring a model with better memory.

In light of this, the scope of our investigation becomes:

Comparing and benchmarking the effectiveness of RNN- and RC models for experimental time series prediction, aiming to contrast the dynamical memory of the models and draw conclusions about what type of datasets they favour from.

The weights of a VRNN are explicitly trained; hence, it is reasonable to believe that the nature of its memory can be tailored to specific situations. On the other hand, there is no reason to believe that RC has this ability. This study aims to investigate whether VRNN has a flexible memory, which influences its performance in a meaningful way.

Theoretical understanding is of the essence, when drawing conclusions from test results. Relevant background information is presented below.

A. Artificial Neural Networks

An artificial neural network (ANN) is a type of machine learning algorithm, which was originally developed to simulate the basic neural information processing in the brain [16]. Indeed, the structure of an artificial neural network clearly reminds of a biological neural network, as regards the aspects brought up at the beginning of this paper. An ANN is a network of connected artificial neurons. An artificial neuron is a simple mathematical function that takes linear combinations of inputs and returns a certain activation value, which can be passed on to other neurons [17]. To get the activation of a neuron, a *neuron activation function* is applied to the linear combination of inputs, which enables the model to learn non-linear and complex patterns [18]. To each of the network's connections corresponds a certain *weight*, modulating the effect of the signal being sent through that connection. The system learns a certain task by fine-tuning the parameters, i.e. setting the weights and biases of linear combinations [19]. The most common and successful approach to training artificial neural networks is the *back-propagation* (BP) algorithm. In this setup, an error gradient propagates backwards through the network, determining which types of weight changes that most efficiently reduce an error within the given task [6]. This algorithm, however, is commonly viewed as being biologically implausible [6] [20]. The most standard category of ANNs are feedforward neural networks. In this structure, information signals is simply communicated in one direction, wherefore the output signal of the system is completely determined by a given input signal and the history of input signals and network states is not considered [21]. This results in

a lack of memory within the system, which is intuitively disadvantageous, in the context of time series forecasting, since the task amounts to predicting future output based on historical input. A contrasting approach is summarised in [21]: "To describe dynamical processes that depend on inputs and its own history, a neural network model must have recurrent connections through which a network activity in the past influences its own activity at present". This is the underlying idea of a recurrent neural network; in general, any neuron may be connected to any other, but the characteristic features is that the network possesses cycles [14]. Since the network state depends on historical input, we say that the network has *dynamical memory* [14]. When trained properly, a recurrent neural network is a powerful tool for prediction of long range temporal data [12]. This is exemplified by the result in [22], [23], [24] and [25]. Importantly, "[RNNs] are the Machine Learning (ML) model most closely resembling biological brains" [13]. The structure of this type of network is illustrated in Figure 1.

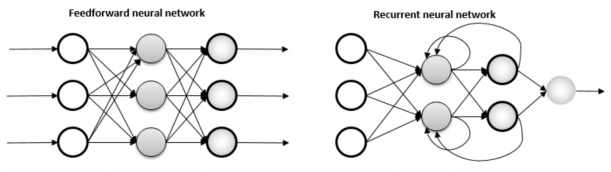


Fig. 1. Artificial Neural Networks Architectures [26]

B. Vanilla Recurrent Neural Network

Exactly how to use utilise past states for future predictions is an important question, within the field of biological learning [27] [28]. This is known as the problem of temporal credit assignment (TCA) [28]. Successfully training a recurrent neural network would give a solution to this fundamental problem, making the model a promising machine learning approach when it comes to time series prediction [29]. However, the temporal aspect of a recurrent neural network, means that we need to extend the back-propagation algorithm in order to apply it. The resulting solution is known as *back-propagation through time* (BPTT) [12]. A recurrent neural network, which is trained via BPTT, is referred to here as a vanilla recurrent neural network (VRNN). Although, the recurrent nature of the network is brain-like, the use of a biologically implausible back-propagation algorithm makes the VRNN model fundamentally different from the brain. The following description of VRNNs is based on the contents of [12] and the complete structure is illustrated in Figure 2. A VRNN has three layers: *input layer, hidden layer* and *output layer*, all comprised of neurons. The layers are connected to each other via weights, jointly represented by matrices; $\mathbf{W}^{\text{in}}$ contains the connections between input- and hidden layer, while $\mathbf{W}^{\text{out}}$ contains connections from hidden- to output layer. Moreover, neurons in the hidden layer are connected to each other via a matrix of recurrent weights denoted by $\mathbf{W}$ and $\mathbf{x}(t)$ denotes the activation values of these neurons at discrete time step t. The following update rule is applied within the hidden layer:

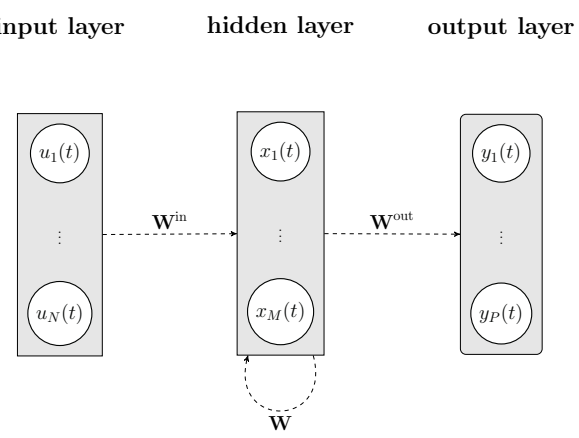


Fig. 2. Vanilla Recurrent Neural Network Architecture: N, M and P are arbitrary positive integers

$$\mathbf{x}(t) = f_H(\mathbf{W}^{\text{in}}\mathbf{u}(t) + \mathbf{W}\mathbf{x}(t-1) + \mathbf{b}_H)$$

where $\mathbf{u}(t)$ is the input signal at discrete time step t. Furthermore, f_H is a neuron activation function, applied element-wise and $\mathbf{b}_H$ is a vector of biases. The corresponding output vector is given by

$$\mathbf{y}(t) = f_O(\mathbf{W}^{\text{out}}\mathbf{x}(t) + \mathbf{b}_O)$$

where f_O is once again a neuron activation function and $\mathbf{b}_O$ is a bias-vector. BPTT can be seen as unfolding the recurrent network and propagating error gradients backward like regular BP. Although this approach solves the TCA problem, back-propagation through time has complications of its own [12]. As the gradient propagates back through time, there are two major pitfalls: the gradient may shrink exponentially in magnitude, causing the network to ignore long time dependencies or it may explode, yielding large weights and unstable behaviour. According to [12], RNNs have difficulties utilising long term temporal behaviour when back-propagation through time is used.

C. Reservoir Computing

At the start of the 21st century a new approach was developed. Known today as reservoir computing (RC), this alternative framework utilises recurrent connections, yet is able to circumvents the training difficulties associated with VRNNs [14] [13]. Since the RC model does not use back propagation, it can be considered as brain-inspired. The following description of reservoir computing is based on the contents of [14] and [13]. The basic idea of a reservoir computing model is to create a random, static recurrent neural network – known as a *reservoir* – which is not directly trained. As such, no back-propagation is needed and the model is not subject to vanishing or exploding gradients. Just like VRNNs, the RC model has three layers of neurons – input, reservoir and output – but rather than fine-tuning all the weights of the network, only the *readout layer* is trained. More specifically, the activations in the output layer is a linear combination of reservoir activations and nothing more than the weights of this combination is

learned. Formally, the output vector $\mathbf{y}(t)$ at a discrete time step t is defined by

$$\mathbf{y}(t) = \mathbf{W}^{\text{out}} \begin{bmatrix} 1 \\ \mathbf{u}(t) \\ \mathbf{x}(t) \end{bmatrix} \tag{1}$$

Here, $\mathbf{u}(t)$ is an input signal and $\mathbf{x}(t)$ is the reservoir activation values, all at time step t. Importantly, $\mathbf{W}^{\text{out}}$ is an output weight matrix, also known as the readout layer – the entity to be trained. Furthermore, the reservoir state is updated according to the following:

$$\tilde{\mathbf{x}}(t) = f\left(\mathbf{W}^{\text{in}} \begin{bmatrix} 1 \\ \mathbf{u}(t) \end{bmatrix} + \mathbf{W}\mathbf{x}(t-1)\right)$$

$$\mathbf{x}(t) = (1-\alpha)\mathbf{x}(t-1) + \alpha\tilde{\mathbf{x}}(t)$$

where $\mathbf{W}^{\text{in}}$ is a matrix containing weights of connections between input layer and reservoir, while $\mathbf{W}$ is a matrix containing weights of reservoir connections. Both matrices are generated randomly and are not subject to training. Moreover, $\tilde{\mathbf{x}}(t)$ corresponds to the update of neuron activations and the *leaking rate* α controls the rate at which the network updates. The neuron activation function f is applied element-wise. The update equation contains an implicit bias vector, namely the first column of $\mathbf{W}^{\text{in}}$. The complete model – comprised of input layer, output layer and reservoir – may also be referred to as an *Echo State Network* (ESN). As an example, assume that we wish to predict a three-dimensional time series dataset. In this case $\mathbf{u}(t) \in \mathbb{R}^3$, is the data at discrete time t and $\mathbf{y}(t) \in \mathbb{R}^3$ is the corresponding one-step-ahead prediction, i.e. $\mathbf{y}(t)$ is a prediction of $\mathbf{u}(t+1)$. The RC model corresponding to our example is illustrated in Figure 3.

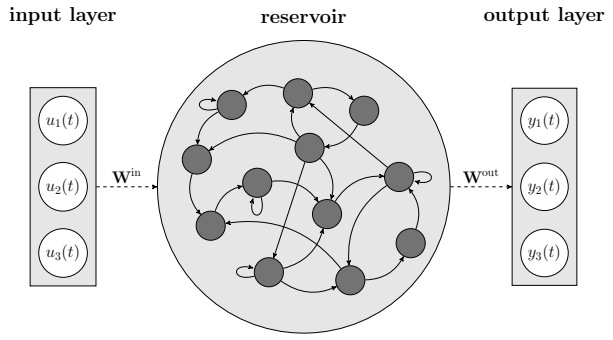


Fig. 3. Reservoir Computing Architecture

Assume that we are given a training set of input vectors $\mathbf{u}(t) \in \mathbb{R}^N$ and target output vectors $\mathbf{y}^{\text{target}}(t) \in \mathbb{R}^N$ for discrete time steps $t = 1, \ldots, T$. The aim of an RC model is to learn the output weights which minimise an error measure given by

$$\frac{1}{N}\sum_{i=1}^{N}\sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_i(t) - y_i^{\text{target}}(t))^2}$$

i.e. the average *Root-Mean-Square Error* ($RMSE$) over all dimensions. Furthermore, as mentioned in [13]: "Extremely large $\mathbf{W}^{\text{out}}$ values may be an indication of a very sensitive and unstable solution". In other words, there is a bias-variance trade of, urging us to penalise large magnitudes of the output weights. The standard method for solving such a problem is to use *ridge regression* and the corresponding matrix solution is found via

$$\mathbf{W}^{\text{out}} = \mathbf{Y}^{\text{target}}\mathbf{X}^{\top}(\mathbf{X}\mathbf{X}^{\top} + \beta\mathbf{I})^{-1} \tag{2}$$

where

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}(1) & \ldots & \mathbf{y}(T) \end{bmatrix},$$
$$\mathbf{X} = \begin{bmatrix} 1 & \ldots & 1 \\ \mathbf{u}(1) & \ldots & \mathbf{u}(T) \\ \mathbf{x}(1) & \ldots & \mathbf{x}(T) \end{bmatrix}$$

and $\mathbf{I}$ is the identity matrix with appropriate dimensions. Finally, β is a *regularization parameter*, modulating the significance of the latter objective (small weights) in relation to the former (minimal error). Generally speaking, a larger value of this parameter reduces variance but increases the bias of a model. Training the readout layer like this, using a pre-existing training dataset in order to fit the output weights in accordance with (2) is referred to as *offline training*, in the context of reservoir computing.

Even though the RC framework might seem relatively simple – being trained via ridge regression, which is rudimentary in comparison to back-propagation – it comprises a productive and promising research field [13] [14]. Firstly, the model has had practical success, performing excellently on benchmarks tasks and clearly outperforming similar methods in chaotic time series prediction [14], as exemplified by [30]. Secondly, the method shows theoretical promise as it (in principle) is capable of simulating any system with continuous output, updating in continuous time, within reasonable constraints on resources and computational time [14] [31]. Thirdly, the RC framework is not subject to *catastrophic inference* (i.e. "the loss or disruption of previously learned information when new information is learned" [32]), since new information can be learned by simply training a new readout layer, while saving the old ones that correspond to a previously learned task [14]. Lastly, RC is biologically plausible, having a plethora of structural similarities with the brain. It is even stated in [14] that RC "provides explanations of why biological brains can carry out accurate computations with an "inaccurate" and noisy physical substrate".

There are, however, problems with the presented model, when it comes to utilising dynamical memory. Assume that we are given a one-dimensional time series dataset $u(t) \in \mathbb{R}$. Further, assume that we wish to predict the time series at discrete time step $t+1$, based on a window of information of size $N+1$, i.e. $[u(t-N), \ldots, u(t)] =: \hat{\mathbf{w}}$. In the classical RC approach (described above), the inputs would be fed iterative to the one input node, updating the network states, before making the prediction $y(t)$ of $u(t+1)$. Note here, that the final prediction is only based on one input sample, namely $u(t)$ and the historical information is only utilised via reservoir state updates. This relates to the so called *echo state property* of RC models – meaning that activation states are uniquely defined by input history, where more significance is given to recent

signals – which is needed for the setup to work properly [13]. Looking at equation (1), it might however be advantageous to simultaneously input all of $\hat{\mathbf{w}}$, in order to more directly utilise the available information for forecasting. This approach would require an input layer with $N+1$ nodes, taking $\hat{\mathbf{w}}$ as input. The corresponding output would be a scalar value approximating $u(t+1)$. Since this model takes time lagged time series values as input, this type of model is referred to here as a time lagged reservoir computing (TLRC) model. The architecture of this model is illustrated (for the one-dimensional case) in Figure 4.

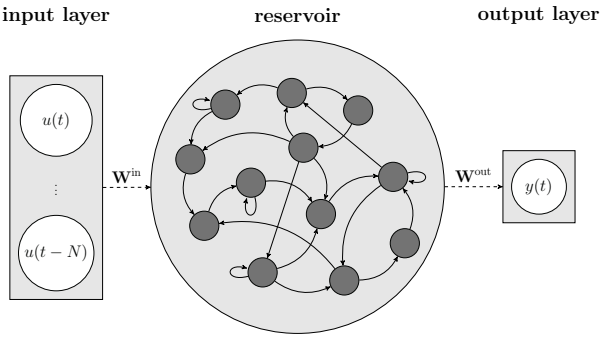


Fig. 4. Time Lagged Reservoir Computing Architecture: window size $N+1$

It is unclear how to implement an analogous model in the multivariate case. In our study, TLRC is used only in the context of univariate time series forecasting. How to extend the model for multiple dimensions is left to future research within the field.

D. Autoregressive Integrated Moving Average

Autoregressive Integrated Moving Average (ARIMA) is a model, which is able to accurately forecast univariate time series, by utilising dependencies in historical information, removing time-dependencies via differencing and correlating observations to residual terms by considering a moving average [33]. The approach is well understood and has been used to represent the body of traditional forecasting models [33]. Owing to its reliability, the model is used here as a way of generating benchmark results, providing a standard for comparison. The mathematical formula is given by

$$(1-\sum_{i=1}^{p}\phi_i L^i)(1-L)^d y_t = (1+\sum_{j=1}^{q}\theta_j L^j)\epsilon_t$$

where p is the order of the autoregressive (AR) term, d is the degree of differencing (i.e., the number of times the data is differenced to make it stationary), q is the order of the moving average (MA) term. Furthermore ϕ_i and θ_j are the AR and MA coefficients, respectively. L is the lag operator, defined as $L^i y_t = y_{t-i}$, y_t is the time series and ϵ_t is the white noise error term at time t.

II. Method

When generating results, we proceed in accordance with the purpose of our research; the idea is to make predictions of time series datasets, while varying the history of information available to the respective models. The amount of time steps that a model bases its final prediction on is referred to as *window size* going forward. In Figure 5 this value corresponds to the amount of discrete time steps in the window of information.

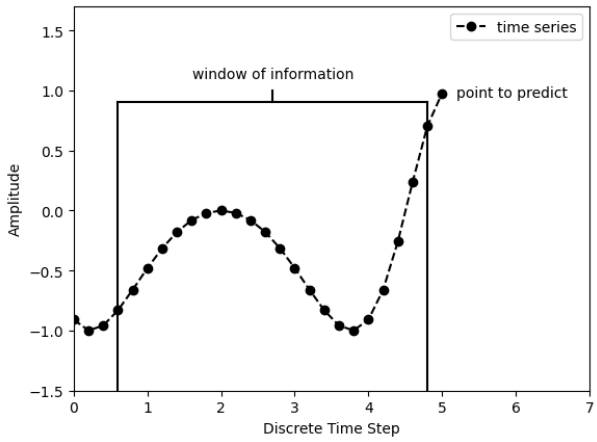


Fig. 5. A visualisation of the window of information inputted to a model for prediction. The *window size* corresponds to the number of points in the window of information.

In order to get more general results, and since window size goes hand in hand with historical dependencies, we perform our test on numerous datasets of different natures. Each dataset contains 1000 samples and are forecasted using one-step-ahead prediction.

Another important parameter considered here is *network size*, referring to the number of neurons in the reservoir in the case of RC models, and to the numbers of neurons in the hidden layer in the case of VRNN. According to [13] a bigger reservoir generally leads to better performance, but also increases computational cost. For this reason, we will not be tuning the network size like other hyperparameters, but rather retest our results using differing network sizes, to see how the outcome depends on this factor.

A compilation of the considered aspects, as well as how they are chosen, is listed below.

Hyperparameters:

- window size: chosen in relation to time scale correlations
- network size: 10 to 460 (increment of 50)

To evaluate these parameters we will perform three types of tests for each dataset. Results for different datasets are presented independently. The performed tests are listed and explained in detail below:

Tests:

1. We forecast the datasets, considering the effect of different window sizes. The autocorrelation function of a dataset is plotted for the first 1000 time steps. The result is discussed, and suitable window sizes are chosen based on the time scale correlations. This test compares model performance, as well as investigating whether performance is related to autocorrelation. The results for this test is presented in tables displaying the average R^2-*score* and $RMSE$ as well as the standard deviation of the

$RMSE$, for prediction using each window size. When possible, the number of iterations to take an average over is chosen such that the standard deviation becomes low in relation to the measurement. In each table, we highlight the window size corresponding to the best accuracy ($RMSE$) of each respective method. The most accurate performances are compared using bar-charts. If any interesting relations are found between performance and autocorrelation, these are furthermore presented and discussed. The window size is set to 100 if the number of independent real values inputted is less than 100, and equal to this number otherwise.

2. We take network size into account, by varying this parameter while testing each model using the best respective window size obtained in the previous test. The result is presented using bar charts.

The remaining part of this section gives a technical description of specific methods used in this work, as well as their implementations. All presented methods were implemented using `Python 3.10`, trained using 90% of the dataset and tested on the remaining 10%. To facilitate an easier read, we let the dimension of a time series be denoted by N.

A. Implementation of ARIMA

ARIMA was implemented using `statsmodels` module and sub-library `tsa.arima.model.ARIMA` that provides an interface for different ARIMA-type models. They provide functions for both training and predicting, which are used to generate benchmark results. Hyperparameters are found using a simple grid search. The ARIMA model can only handle univariate time series, so in the case of multivariate datasets different dimensions are forecasted independent of each other.

B. Implementation of Vanilla Recurrent Neural Network

For the implementation of the VRNN we use the Python library `Tensorflow` and more specifically the `keras` API [34]. An input layer, of shape

$$\texttt{(window size, N)}$$

is used to feed the input time series to the hidden layer, performing the RNN operations. The hidden layer is connected to an output layer, of size

$$\texttt{(1, window size)}.$$

The model uses the activation function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

for both the hidden layer and the output layer.

The hyperparameters play a vital role in the performance of the model and choosing good hyperparameters is thus crucial for a fair comparison. To do this we tune all hyperparameters in [34] (excluding network size and window size) via an *Adam optimiser* – a stochastic gradient descent method.The model is trained for 15 epochs via BPTT. After training, we iteratively predict the remaining time series using the specified window of historical data.

C. Implementation of Reservoir Computing Model

The RC model is implemented using the a Python library `ReservoirPy`. The model consists of two main components, namely the reservoir and the readout, explained above. In our implementation, the reservoir consists of *n* randomly connected neurons, and the number of input nodes is N. The connections are not trained and instead randomly initialised depending on several hyperparameters. The activation function $\tanh$ is then applied on the internal states. The readout consists of a single layer of nodes, that learns using ridge regression [35], as discussed in the background. In the context of reservoir computing both the leaking rate α and the *spectral radius* ρ are essential hyperparameters [13]. The leaking rate regulates how much the network's former state influences its current state, while the spectral radius modulates the scale of reservoir weights [13]. For suitable choices of ρ, the echo state property is satisfied [13]. To optimise these hyperparameters, we employ a *random search* algorithm, which selects values randomly from a preset range and evaluates them. Other than window size and network size, the remaining hyperparameters are set to their default values in [35]. The model predicts iteratively using one-step-ahead prediction.

In addition to the standard RC, we implemented a time lagged version of the model. To achieve this we applied a sliding window operation to the time series data, utilising a window of a specific size, where subsequent windows differ by one time step. This produces a windowed view of the data, which is reorganised into a `2D` array format, where each row represent a window of discrete time steps. As a result, we are able to match the input dimension to the window size and feed the data into the RC model. The model was then trained via matching the row to the corresponding following time step and tuned in a similar fashion to the standard version of RC. Note that this approach only works for the univariate case.

D. Evaluation/Measurements

The described models are implemented as `Python` functions, taking test- and train datasets as input and returning two metrics: Root-Mean-Squared Error ($RMSE$) and R^2-score. $RMSE$ measures the accuracy of the prediction and is given by

$$\texttt{RMSE} = \sqrt{\frac{1}{T}\sum_{i=1}^{N}\sum_{t=1}^{T}(y_i(t) - \hat{y}_i(t))^2}$$

where $\hat{y}_i(t)$ is our prediction of the ith attribute $y_i(t)$ of the data, all at discrete time step t. Naturally, T is the length of predicted data. The R^2-score measures the proportion of variation which the model can account for. It is given by

$$\texttt{R}^2 = 1 - \frac{\sum_{i=1}^{T}(y_i(t) - \hat{y}_i)^2}{\sum_{i=1}^{T}(y_i(t) - \bar{y})^2}$$

where $\bar{y}$ is the mean value over all the observed data. These values are used to evaluate the models' performances within various tests. The functions furthermore returns the predicted time steps along with their actual values.

The autocorrelation function was used to analyse the time scale correlation of a dataset. It was implemented using

the `.autocorr()` method from the `Pandas` module. The method returns the so called *Pearson correlation* between a time series and lagged version of itself [36], and the autocorrelation function is generated by applying it iteratively for different lags. The Pearson correlation is a measure of the linear relationship between two datasets X and Y that share attributes [37] [38]. The metric has range $[-1, 1]$ and is given by

$$\mathrm{R}_{\mathrm{xy}} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where x_i and y_i are the attribute values of the datasets and $\bar{x}$ and $\bar{y}$ are their respective means [38]. Hence, the autocorrelation of a time series y at lag k is given by

$$\hat{\rho}(k) = \frac{\sum_{t=k+1}^{T}(y(t) - \bar{y})(y(t-k) - \bar{y})}{\sqrt{\sum_{t=k+1}^{T}(y(t) - \bar{y})^2}\sqrt{\sum_{t=k+1}^{T}(y(t-k) - \bar{y})^2}}.$$

Time scale correlation is used in this research as a proxy for periodicity. The autocorrelation of a time series is high if the lag is close to a period length [9], but a time series may have peaks in its autocorrelation function without being periodic or seasonal. Furthermore, the autocorrelation function may have different types of peaks, appearing with different amplitudes or frequencies. This corresponds to dependencies at different scales, e.g. daily or monthly. A concept, discussed in the *Result* section is the average autocorrelation corresponding to a certain window size. Assume we want to predict a time series y at time step t. If the window size is N, this means that we are given the window of information $[y(t-N), \ldots, y(t-1)]$. We define the average autocorrelation for that window size (N) to be

$$\bar{\rho}(N) = \frac{1}{N}\sum_{k=1}^{N}\rho(k)$$

that is, the average correlation over the lags for which the time series is known. Since the autocorrelation conveys how much a time-series corresponds to itself at a certain lag, this value would entail the average dependency between the available information and the value we wish to predict. Our hypothesis is that this value will influence a models understanding of a time series when it is given a certain window of information.

III. Datasets

The datsets were divided into two main types: *benchmark data* and *experimental data*. The first part of this section gives a comprehensive overview of the data used for benchmark tests, evaluating the networks' performances within the tasks detailed in the previous section. Likewise, the second half of this section surveys the experimental datasets used for testing. Another key aspect of the time series, taken into consideration here, is whether they are univariate or multivariate, meaning that their dimension is respectively one or more.

A. Benchmark Datasets

Two standard datasets were used for benchmark testing and comparisons.

Mackey-Glass Time Series: The *Mackey-Glass* time series is a commonly used benchmark for univariate time series forecasting [15], especially in the context of chaotic time series. The dataset is obtained by solving the Mackey-Glass time delayed differential equation.

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1 + x(t-\tau)^n} - bx(t) \tag{3}$$

where $a = 0.2$, $b = 1.0$, $\tau = 17$ and $n = 10$ are constant parameters of the equations as in [35]. For this paper, the `mackey_glass` dataset implemented in the `ReservoirPy` library was utilised. The library derives the time series by solving (3) on a continuous space using a 4-5th order Runge-Kutta [35] and returns the time series for a specified number of time steps. The time series is illustrated in Figure 6.

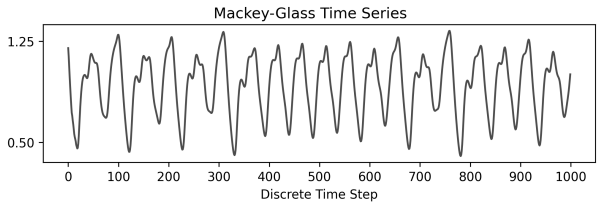


Fig. 6. Mackey-Glass Time Series

Lorenz Time Series: As a benchmark for multivariate predictions, the *Lorenz* time series dataset is used. It was first defined by Edward Lorenz, and is obtained by solving the system of ordinary differential equations [35]:

$$\begin{aligned}\frac{dx(t)}{dt} &= \sigma(y(t) - x(t)) \\ \frac{dy(t)}{dt} &= x(\rho - z) - y \\ \frac{dz(t)}{dt} &= xy - \beta z\end{aligned} \tag{4}$$

where $\rho = 28.0$, $\sigma = 10.0$ and $\beta = 8/3$ are constant parameters of the system. The `lorenz` dataset from `ReservoirPy` library solves (4) and returns a dataset over n continuous time steps. The equation (4) is notoriously known for its chaotic behaviour. The amplitudes of the variables are 3D-plotted in Figure 7 and graphed separately in Figure 8.

B. Experimental datasets

Three experimental datasets are used to generate results for our models. These time series pose a greater challenge to accurately predict and has more practical relevance than the ones described previously. To get faster convergence and more accurate predictions we normalise datasets of this type, in order to "minimise the effect of features with higher numerical contributions than others" [39]. We used Min–Max Normalisation to the range $[0, 1]$, as described in [40]. In the used formula, a normalised data point is given by

$$y_i'(t) = \frac{y_i(t) - y_i^{\min}}{y_i^{\max} - y_i^{\min}}$$

where $y_i(t)$ is the ith attribute of the data at discrete time step t, while $y_i^{\min}$ and $y_i^{\max}$ respectively denotes the minimal

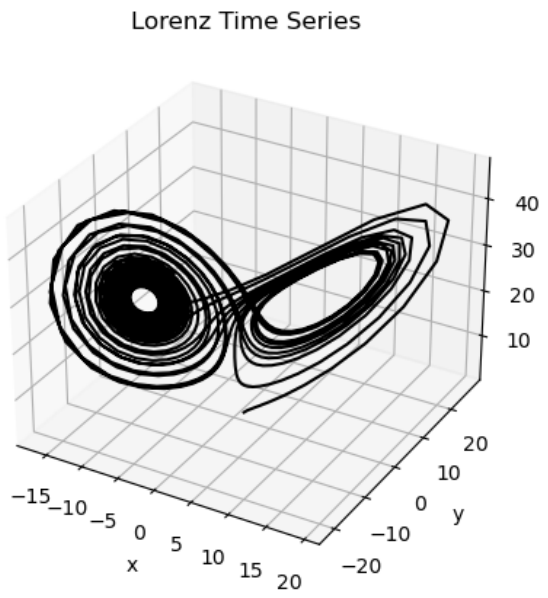


Fig. 7. Lorenz time series: 3D Plot of 1000 Time Steps

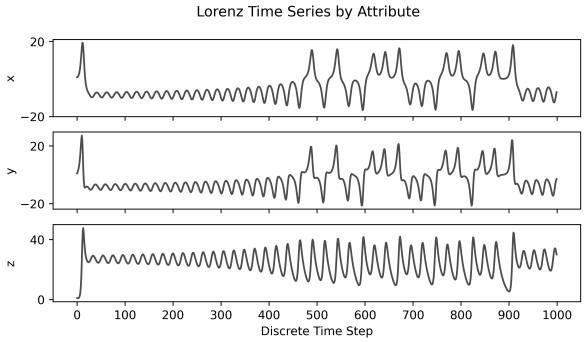


Fig. 8. Lorenz Time Series: Separated Variables

and maximal value of this attribute over all time steps. All experimental datasets are described below.

OMXS30 Closing Price: This time series contains the closing price of the OMXS30 index, dating back 1000 business days up until 04/04-2023. The time series is shown in Figure 9 and was retrieved from Yahoo Finance.

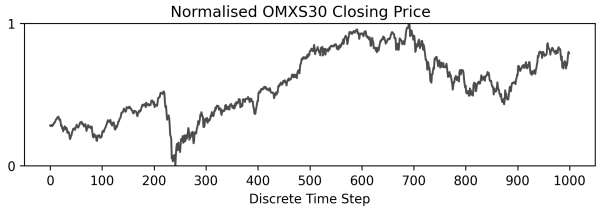


Fig. 9. Normalised Closing Price of the OMXS30 Index

Average Daily Temperature in Stockholm: This time series contains the average air temperature in Stockholm during the time period between 09/09-1932 and 05/06-1935, retrieved from [41]. The dataset was chosen due to the clear seasonality of weather patterns. It is shown in Figure 10.

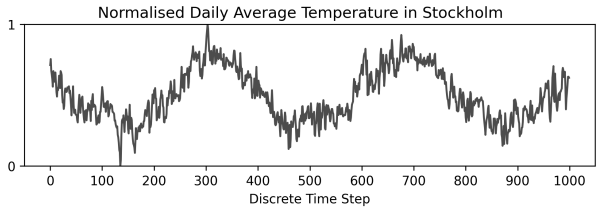


Fig. 10. Normalised Average Air Temperature per Day

Bonn EEG Recordings: The final experimental dataset consists of three electrodes from an EEG recordings of a single healthy patient from [42], making it the only multivariate experimental time series in our study. The electrodes were shorted to cover 1000 time steps and selected randomly. Using a 128-channel acquisition system, the EEG data were collected at a frequency of 173.61 Hz and are shown in Figure 11.

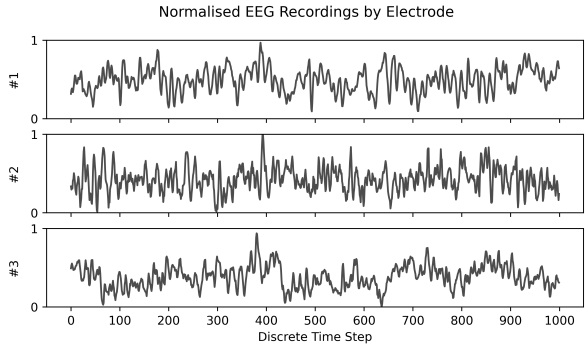


Fig. 11. EEG Recordings From 3 Electrodes

IV. Results

A. Mackey-Glass Test Results

Mackey-Glass Test 1: The autocorrelation of the Mackey-Glass time-series is displayed in Figure 12. The data is highly periodic, with one period corresponding to around 50 time steps. The chosen window sizes are thus in the range $[1, 50]$.

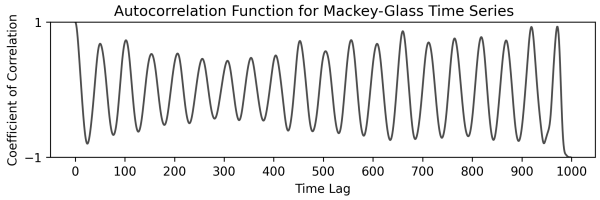


Fig. 12. Autocorrelation function for 1000 time steps of the Mackey-Glass time series. The data is highly periodic with a period of circa 50 time steps.

Displayed in Figure 13, is the lowest obtained $RMSEs$, when varying the window size. As seen there, VRNN is performing worse than ARIMA in terms of accuracy. Furthermore, the RC models are performing distinctly better than both of the others. Notably, TLRC has the best performance within both $RMSE$ and R^2-score, as seen in Table I in the appendix. All models (including ARIMA) are sensitive to window size,

but the performances are generally good. Recall however that the dynamics of the Mackey-Glass time series are quite simple.

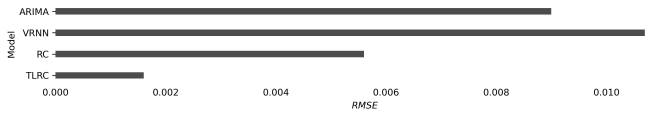


Fig. 13. The lowest obtained $RMSE$ for each model when varying the window size used for prediction of the Mackey-Glass time series.

An interesting correspondence can also be observed between the autocorrelation of the time series and the performance of the RC model. As shown in Figure 14, the R^2-score achieved by RC for a certain window size seems to be roughly proportional to the average autocorrelation corresponding to that window size. This result could be reasonable in general. Recall that the R^2-score measures the proportion of variation that can be predicted by the model, while autocorrelation indicates how these variations depend on time series history. If a model is capable of harnessing the dependencies within the information it is given, its understanding of the variations (R^2-score) should depend on dependencies in its input (autocorrelation). A similar result was not observed for the remaining models, which roughly seemed to get better with larger windows.

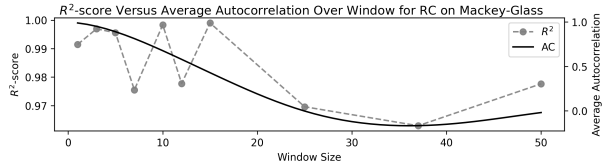


Fig. 14. The R^2-score achieved by RC, when using a certain window size (on the primary y-axis), plotted against the average autocorrelation of the Mackey-Glass time series for the window size (plotted on the secondary y-axis). The Pearson Correlation between the plotted lines is 0.71

Mackey Glass Test 2: Figure 15 shows the resulting $RMSE$ for different models when predicting the Mackey-Glass time series with varying network sizes. VRNN performance pummels for larger sizes (perhaps due to overfitting). RC and TLRC, on the other hand, works fine with larger reservoirs and the performance seems quite stable over window sizes.

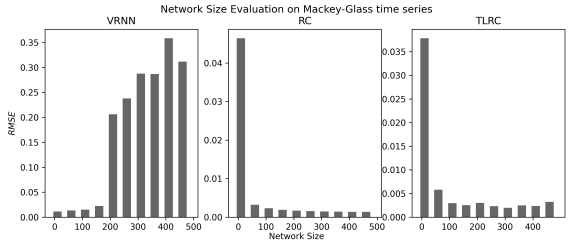


Fig. 15. The achieved $RMSE$ for each model on the Mackey-Glass time series, when using the best respective window size from test 1 (see table I in appendix) and varying the network size.

B. Lorenz Test Results

Lorenz Test 1: The autocorrelation function of the Lorenz time series is shown in Figure 16. The plot contains patterns of peaks and valleys, that may resemble periodicity. The patterns are not truly periodic though, and reflects the chaotic nature of the time-series. As mentioned in [43], the Lorenz time-series displays a quasi-periodic structure. Indeed, the z-coefficient shows quite periodic behaviour, but the x- and y-coefficients do not. Moreover, autocorrelation in the z-attribute has larger magnitude, than the others, which fade quickly and only show faint peaks. For this reason, we will choose window sizes mostly in relation to the z-attribute. A period in this dimension has a length of around 25 time steps, so we choose points mainly in the range of $[1, 30]$. Furthermore, larger window sizes may provide valuable insights and the x- and y-coefficients have a small peak at about $t = 130$, prompting us to also test 100 and 130.

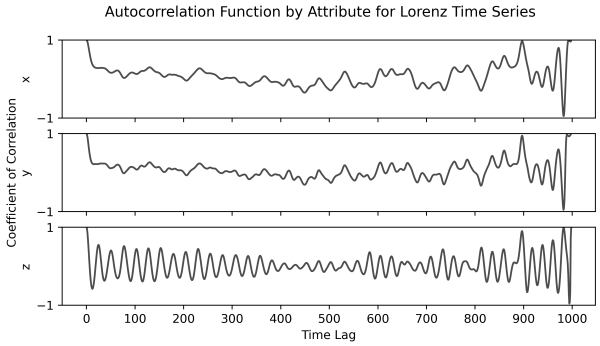


Fig. 16. Autocorrelation function for 1000 times steps of the Lorenz time series. Although the data is chaotic, the z-attribute has a periodic nature with a period of circa 25 time steps.

Looking at Table II in the appendix, an interesting observation is that ARIMA underperforms starkly when the window size is low. This indicates that the time series is complex and cannot easily be forecasted – in part due to its multivariate nature. Furthermore, the performance of VRNN generally seems to get better with larger window sizes, achieving its best performance for window size 100. As seen in Figure 17, the performance ranking (VRNN, ARIMA, RC) agree with the Mackey-Glass case and the advantage of RC is even clearer here. It is worth emphasising that the ARIMA model used here was not developed for multivariate time series prediction, handling the dimensions separately, but outperforms VRNN nonetheless.

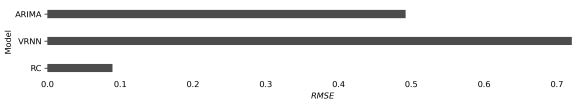


Fig. 17. The lowest obtained $RMSE$ for each model when varying the window size used for prediction of the Lorenz time series.

The relation discussed in *Mackey-Glass Test 1* could not be identified here, partly because analysis of autocorrelation is more difficult in the multivariate case.

Lorenz Test 2: Results found in Figure 18 indicate that VRNN once again disbenefits from larger network sizes, while the performance of RC improves. This result agrees completely with the Mackey-Glass case. Moreover, RC again exhibits stable behaviour.

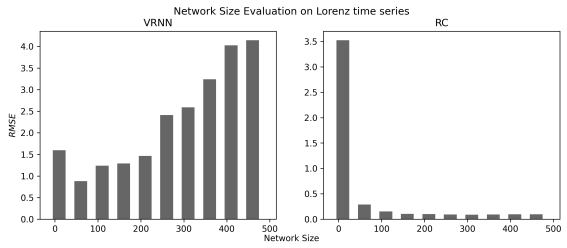


Fig. 18. The achieved $RMSE$ for each model on the Lorenz time series, when using the best respective window size from test 1 (see Table II in appendix) and varying the network size.

C. OMXS30 Test Results

OMXS30 Test 1: The autocorrelation function for the OMXS30 index is shown in Figure 19. Although there are no clear periodic patterns, approximately the first 300 time steps exhibit correlation. Further, there is a sharp peak just before 800 and a potentially interesting valley around 670 time steps.

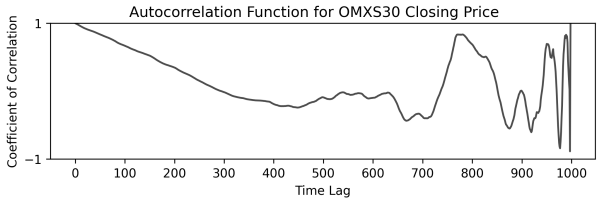


Fig. 19. Autocorrelation function for 1000 times steps of the OMXS30 time series. The data is not periodic, but has some interesting peaks and valleys.

When predicting the OMXS30 time series ARIMA outperforms RC for the first time as seen in figure 20, suggesting that RNN- and RC models might struggle in practical and complex situations, perhaps needing further tuning (discussed partly in *Test 2*). It is also clear from Figure 20 that ARIMA, RC and TLRC perform similarly, but the time lagged model once again has an edge. The perhaps most notabe result in Table III in the appendix is that the R^2-score of VRNN and TLRC became negative for large window sizes and the VRNN forecast could not even be executed for window size 780, indicating that these models are unstable on a large timescale. On the contrary, RC performs remarkably consistently and might even be able to benefit from very large windows, given the right situation.

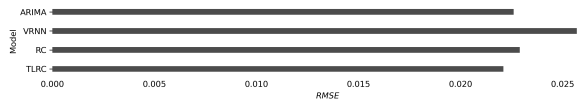
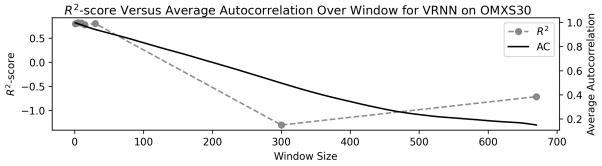
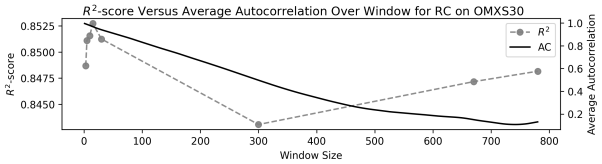


Fig. 20. The lowest obtained $RMSE$ for each model when varying the window size used for prediction of the OMXS30 time series. For this dataset TLRC achieves the lowest error.

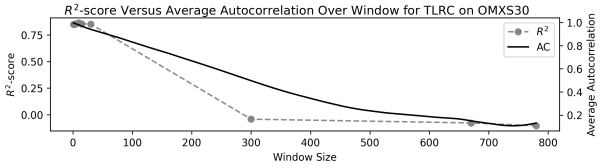
If we disregard window size 1 for RC, which seems to be an outlier in Table III (also having high standard deviation), we see a similar pattern as the one identified in *Mackey Glass Test 1*. As shown in Figure 21b, the correspondence occurs on a large scale, but is not completely convincing. Interestingly, in Figure 21a and 21c, the same pattern seems to be present for VRNN and TLRC, but to a larger extent. In these cases, we do not need to exclude any point[1], and the Pearson correlation is much higher than for RC, especially in the case of TLRC. Moreover, the R^2-score is quite close to the autocorrelation value, again, especially for TLRC. Note that the behaviour is only exhibited when a large time scale is considered – perhaps because the autocorrelation function only has large scale pattern. For TLRC and VRNN it may however be the case that performance diminishes with large window sizes, which just happens to coincide with the decline of the autocorrelation function.



(a) VRNN: The Pearson Correlation between the plotted lines is 0.885



(b) RC: The Pearson Correlation between the plotted lines is 0.641



(c) TLRC: The Pearson Correlation between the plotted lines is 0.968

Fig. 21. The R^2-score achieved by different models, when using a certain window size (on the primary y-axis), plotted against the average autocorrelation of the OMXS30 time series for that window size (plotted on the secondary y-axis).

OMXS30 Test 2: The results found in Figure 22 agree with previous findings, in the sense that VRNN disbenefits from large window sizes, while RC and TLRC are more stable. In this case, RC and TLRC perform remarkably robustly; their accuracy is more or less unaffected by the windows size.

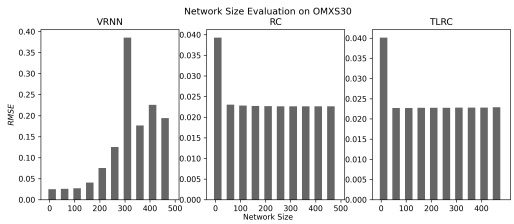


Fig. 22. The achieved $RMSE$ for each model, when using the best respective window size in test 1 (see Table III in appendix) and varying the network size.

[1]Apart from window size 780 for VRNN, since this yielded no result.

D. Average Temperature Test Results

Average Temperature Test 1: The autocorrelation of the weather data in Figure 23 shows a clear seasonal pattern, with periods corresponding to around a year or 365 time steps (days). It is therefore interesting to investigate window sizes $[90,\ 180,\ 365]$, corresponding to a season, half a year and a year. As usual, we will include a spectrum of smaller sizes, in this case ranging from 1 to 15.

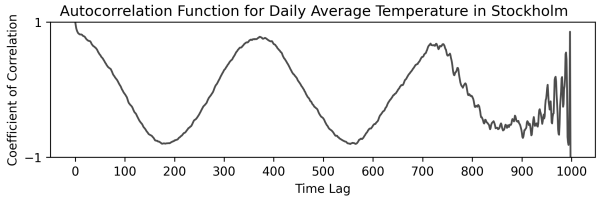


Fig. 23. Autocorrelation function for 1000 times steps of the average temperature time series. The data is periodic with a period corresponding to a year, or 365 time steps.

Looking at the Figure 24, all models are performing quite similarly and in particular, VRNN has less of a disadvantage in this case. Once again, VRNN and TLRC have negative R^2-scores for large window sizes (seen in Table IV in the appendix), confirming their instability. The RC-model, on the other hand, is able to benefit from a large window of information, assuming its greatest accuracy at window sizes 1 and/or 90. In simple terms, RC predicts most accurately when either considering a smaller amount of highly correlated data, or more data which is less correlated on average. It is interesting that a window size of 90 gives better performance than a window size of 180, since $t = 90$ is where the autocorrelation changes sign. Furthermore, this dataset is the first instance at which RC outperforms TLRC, perhaps exactly because this dataset has correlations on a larger time scale. ARIMA seems to harness the periodicity quite well as its performance increases with window size (again seen in Table IV in the appendix) and it achieves the best performance on this dataset, reinforcing the notion that RNN- and RC models disbenefit from experimental data.

Comparing R^2-score with average autocorrelation as before, we again suspect that there is a strong connection between the two, which can be seen in Figure 25. As before, the correspondence is weaker for RC and especially strong for TLRC. Note that we have excluded window size 180 in the case of VRNN, since the corresponding model exhibits highly unstable behaviour in that case. RC demonstrates the behaviour to a smaller extent, but performs much better than the remaining models on the large time scale. Similar to OMXS30 *test 1*, the correlation is only present on a large time scale. As discussed in that section, the apparent pattern might simply be a coincidence in the case of VRNN and TLRC, while this is less probable for RC (since its performance is more stable).

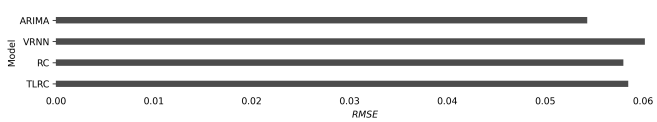
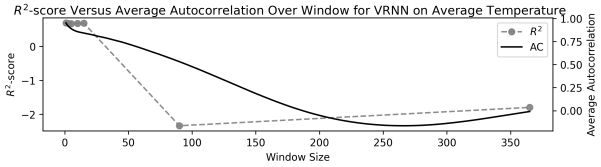
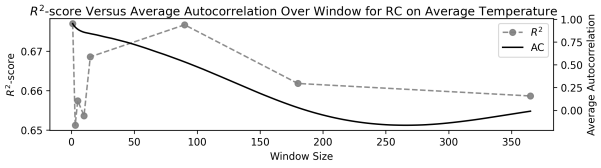


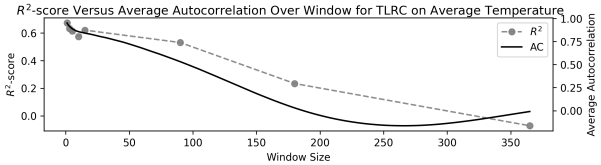
Fig. 24. The lowest obtained $RMSE$ for each model when varying the window size used for prediction of the Average temperature time series. For this dataset ARIMA achieves the lowest error.



(a) VRNN: We have disregarded the window size 180 and the Pearson correlation between the lines are 0.829



(b) RC: No clear correlation is seen



(c) TLRC: The Pearson correlation between the lines are is 0.939

Fig. 25. The R^2-score achieved by different models on the average temperature time series, when using a certain window size (on the primary y-axis), plotted against the average autocorrelation of the average temperature time series for that window size (plotted on the secondary y-axis).

Average Temperature Test 2: As seen in Figure 26 VRNN results are similar to previous findings, where performance declines with network size. For RC and TLRC, the accuracy is once again remarkably constant. This reinforces the belief that reservoir models, and especially RC, are more stable than VRNN. The fact that RC and TLRC perform very steadily on the univariate experimental dataset might however suggest that they are not making predictions by a particularly complex method, otherwise performance might be affected by the amount of neurons, as for VRNN. This would agree with the fact that they are both outperformed by ARIMA and that they perform very similarly.

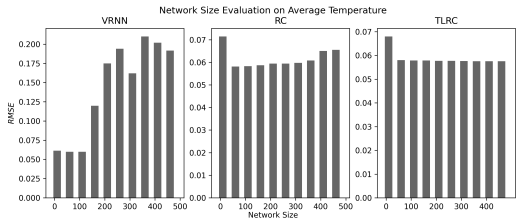


Fig. 26. The achieved $RMSE$ for each model, when using the best respective window size in test 1 (see table IV in appendix) and varying the network size.

E. EEG Data Test Results

EEG Recordings Test 1: The autocorrelation function for the EEG recordings is displayed in Figure 27. Electrode #1 has a small-scale seasonal pattern with behaviours repeating at intervals of about 20 time steps. Furthermore, electrode #3 seems to have an underlying large-scale periodic pattern with first peak at around 150. Although, electrode #2 shows little autocorrelation and an unstructured pattern, it has a first peak at around $t = 20$, which we will take into account. Due to the lack of periodic patterns, miscellaneous window sizes in the range $[1,\ 50]$ were chosen, as well as the larger size 100 and 150.

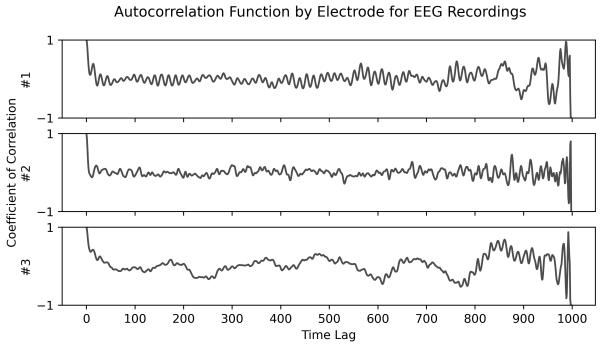


Fig. 27. Autocorrelation function for 1000 times steps of the EEG recordings time series. Although the data is chaotic, the $\#1$-attribute has a small scale periodic pattern, while attribute $\#3$ exhibits some large scale periodicity.

Table V in the appendix shows that this is the first experimental dataset where ARIMA has a clear disadvantage, however this is partly due to the ability of VRNN and RC to encapsulate multiple dimensions. RC once again outperforms VRNN, which is a general pattern over all datasets, but the difference in performance is the smallest in this case, as displayed in Figure 28. We see another example of VRNN being unstable on the large scale, which is also a pattern over all experimental datasets. No connection between R^2-score and autocorrelation could be observed.

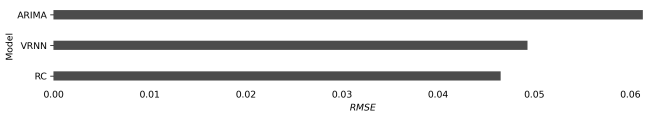


Fig. 28. The lowest obtained $RMSE$ for each model when varying the window size used for prediction of the EEG recording time series. For this dataset RC achieves the lowest error.

EEG Recordings Test 2: Results displayed in Figure 29 show that RC once again is quite unaffected by changes in network size. Moreover, similar to other experimental dataset, the performance could not be significantly improved by altering the network size. This contrasts the Lorenz results, where a larger network size was beneficial.

Summary of Results

A summary of the results can be seen for the univariate case in Figure 30 and for the multivariate datasets in Figure 31.

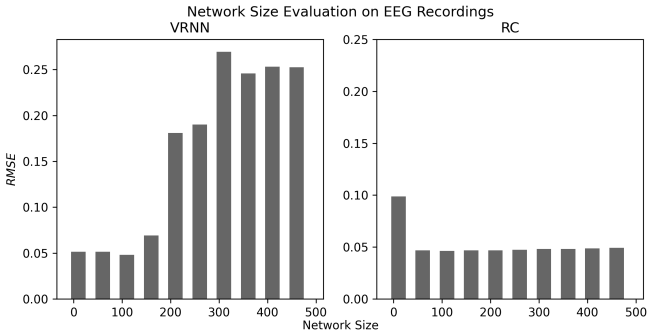


Fig. 29. The achieved $RMSE$ for each model, when using the best respective window size in test 1 (see Table V in appendix) and varying the network size.

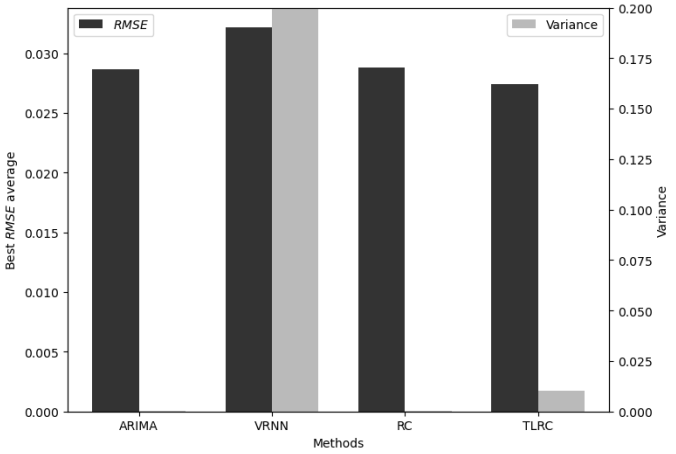


Fig. 30. This bar plot shows the mean of the best achieved RMSE for all datasets (black bars) and the corresponding variance (grey bars) across all window sizes, for four models in the univariate case. Note the low variance for the ARIMA and RC models, as well as the relatively high variance of 0.34 for the VRNN model (not fully visible due to the scale of the y-axis).

V. Discussion/Conclusion

In this paper, we compared four different models, of which two were brain-inspired. The main objective was to compare recurrent neural networks with reservoir computing models. The tests were carried out on five datasets with differing qualities, such as dimensionality and time scale correlations. Two of these also served as benchmarks, while the other three represented a more realistic setting. Despite the variation of tests and datasets, the RC framework consistently outperformed VRNN. Not only did RC achieve better accuracy, but also seemed to exhibit a higher level of stability when considering larger historical horizons. This means that RC handles historical data in a more consistent way. Furthermore we implemented a time-lagged RC model for univariate time series forecasting. TLRC outperformed the classical RC model on two out of the three time series – one benchmark and one experimental. When comparing VRNN and the RC models to the benchmark model ARIMA, we saw that the neural network based models disfavour from practical situations; RC and TLRC are even outperformed by ARIMA on one experimental dataset. Despite disregarding the correlation between dimensions, ARIMA generally outperforms VRNN, with the

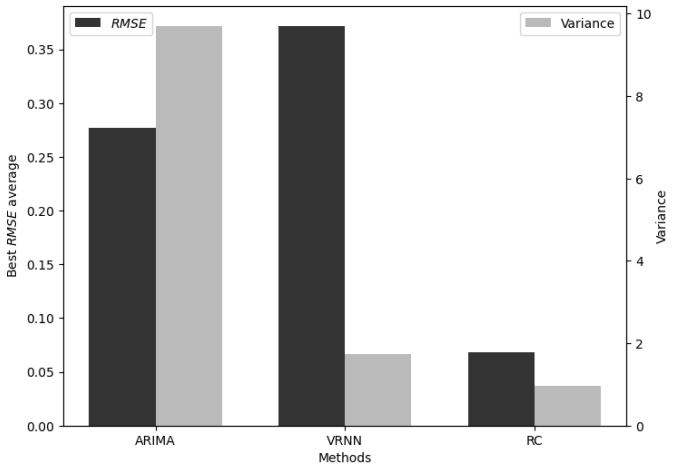


Fig. 31. This bar plot displays the mean of the best achieved RMSE for all datasets (black bars) and the corresponding variance (grey bars) across all window sizes, for three models in the multivariate case. Notably, RC achieves a consistently lower RMSE score than the other models. In contrast, the ARIMA model exhibits a higher degree of variance in its RMSE scores.

only exception being the multivariate EEG-recordings dataset. In order to generate more relevant benchmark results, we suggest that a benchmark model, which is able to more adequately handle multivariate data, is implemented in future research. Note that we have implemented the most rudimentary recurrent neural network model (VRNN) and using a more sophisticated variant (such as LSTM) would naturally improve performances. In addition, brain-inspired models perform similarly on experimental datasets, while their performance differs on simpler time series – in favour of TLRC. In general, TLRC was implemented with success and has potential. Further research could provide better results however, and formulating an analogous model for multivariate forecasting, would be especially interesting.

This study also attempted to asses the memory capabilities of the respective models, via varying the window of information available to them and examining the corresponding results. The basic idea is that a model with better memory should be able to benefit from a larger window. In this respect, RC has the best capabilities. However, having a good memory might also mean that a model is influenced by all the information it is given – in both beneficial or adverse situations. Therefore, we investigated potential relationships between the autocorrelation in the data given to a model and resulting ability of the model to understand the time series. The results indicate that all neural network based models' performances are influenced by the level of autocorrelation in the given data. However, VRNN and TLRC showed a clearer dependency, both in terms of correlation and magnitude; in the considered situations, this meant that the accuracy of TLRC and VRNN decreased drastically with autocorrelation and optimal performances were obtained for small window sizes. However, if the data sets were of a different nature, having large autocorrelation for a longer window of time, the effect might just as well have been the opposite. There are a few possibilities left to explore here, which we leave to future research.

1. The information given to a model need not be connected in time. Instead, one could simply look at the autocorrelation function of a time series and feed in disjoint information windows corresponding to high autocorrelation. VRNN, TLRC and even RC, could benefit a lot from this, if their performance depends much on autocorrelation. This needs to be investigated further.
2. The dynamics of autocorrelation is less straightforward in the multivariate case. This study does not use an adequate tool for analysing autocorrelation in multiple dimensions. No relation between performance and autocorrelation could be found in this case, yet VRNN achieved optimal performance for large window sizes on both multivariate datasets, in contrast to univariate dataset, where the optimal window size was small. Perhaps, the multiple dimensions yield more dependencies within the data, making VRNN effective on a larger scale. Research, investigating multivariate autocorrelation and its relation to time series forecasting is needed to resolve this question.
3. VRNN and TLRC have much larger input layers, than RC. This would naturally make the models more complex, which poses a risk of unpredictable dynamics and overfitting. This might contribute to the instability of the models. This also means that the decrease in accuracy might not be related to autocorrelation, making the above discussion less relevant. Further studies are needed to establish a dependence between model performance and autocorrelation.

Our research also considers the importance of network size when using an appriopriate window size for forecasting, investigating whether optimal results can be improved by varying the network size of a model. The performance of VRNN decreased with large network sizes in all situations. This might be surprising, since more neurons means higher computational cost, but can be explained by overfitting or an abundance of hyperparameters that introduces training difficulties. On the other hand, RC and TLRC was able to benefit from larger network sizes when tested on benchmark datasets. However, in our study, the notion that a larger reservoir should lead to better performance is only true in certain uncomplicated situations. When forecasting experimental datasets, RC and TLRC were largely unaffected by the network size.

A. Validity and reliability of results

To some extent, our study lacks a solid statistical foundation. Complementary statistics, such as hypothesis testing, would help strengthen our conclusion and make the study more complete.

The optimal window size is found in test 1, using a network size of 100 neurons. But the optimality of a window size might be depend on the network size and vice versa. A potential result of this is that network sizes close to 100 are disproportionately advantageous, which affects the results of test 2 and makes them less valid. This would be especially true for VRNN, since this model is the most unstable over both parameters.

ACKNOWLEDGMENT

The authors would like to thank our supervisor Pawel Herman for support, guidance and insightful feedback.

APPENDIX - TEST 1 TABLES

REFERENCES

[1] M. Hofman, "Evolution of the human brain: when bigger is better," *Frontiers in Neuroanatomy*, vol. 8, 2014. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnana.2014.00015

[2] R. C. Fong, W. J. Scheirer, and D. D. Cox, "Using human brain activity to guide machine learning," *Scientific Reports*, vol. 8, no. 1, p. 5397, 2018. [Online]. Available: https://doi.org/10.1038/s41598-018-23618-6

[3] IBM. (2017, Aug.) Brain-inspired ai: How neuroscience helps to advance machine learning. IBM, New York. [Online]. Available: https://www.ibm.com/blogs/research/2017/08/brain-inspired-ai/

[4] L. S. Iliadis, V. Kurkova, and B. Hammer, "Brain-inspired computing and machine learning," *Neural Computing and Applications*, vol. 32, no. 11, pp. 6641–6643, 2020. [Online]. Available: https://doi.org/10.1007/s00521-020-04888-6

[5] Y. Munakata and J. Pfaffly, "Hebbian learning and development," *Developmental Science*, vol. 7, no. 2, pp. 141–148, 2004. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-7687.2004.00331.x

[6] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Reviews Neuroscience*, vol. 21, no. 6, pp. 335–346, 2020. [Online]. Available: https://doi.org/10.1038/s41583-020-0277-3

[7] G. Tiao, "Time series: Arima methods," in *International Encyclopedia of the Social & Behavioral Sciences*, N. J. Smelser and P. B. Baltes, Eds. Oxford: Pergamon, 2001, pp. 15 704–15 709. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0080430767005209

[8] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, "Chapter 8 - data transformations," in *Data Mining (Fourth Edition)*, I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, Eds. Morgan Kaufmann, 2017, pp. 285–334. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128042915000088

[9] E. C. Nwogu, I. S. Iwueze, and V. U. Nlebedim, "Some tests for seasonality in time series data," *Journal of Modern Applied Statistical Methods*, vol. 15, no. 2, p. 24, 2016.

[10] Y. Malode, D. Khadse, and D. Jamthe, "Efficient periodicity mining using circular autocorrelation in time series data," *International Research Journal of Engineering and Technology (IRJET)*, vol. 2, no. 3, pp. 430–436, 2015.

[11] H. Hochheiser and B. Shneiderman, "Interactive exploration of time series data," in *The Craft of Information Visualization*, ser. Interactive Technologies, B. B. BEDERSON and B. SHNEIDERMAN, Eds. San Francisco: Morgan Kaufmann, 2003, pp. 313–315. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781558609150500391

[12] H. Salehinejad, J. Baarbe, S. Sankar, J. Barfett, E. Colak, and S. Valaee, "Recent advances in recurrent neural networks," *CoRR*, vol. abs/1801.01078, 2018. [Online]. Available: http://arxiv.org/abs/1801.01078

[13] M. Lukoševičius, "A practical guide to applying echo state networks," *Neural Networks: Tricks of the Trade: Second Edition*, pp. 659–686, 2012.

[14] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer science review*, vol. 3, no. 3, pp. 127–149, 2009.

[15] S. Shahi, F. H. Fenton, and E. M. Cherry, "Prediction of chaotic time series using recurrent neural networks and reservoir computing techniques: A comparative study," *Machine Learning with Applications*, vol. 8, p. 100300, 2022.

[16] E. Micheli-Tzanakou, "Artificial neural networks: An overview," *Network: Computation in Neural Systems*, vol. 22, no. 1-4, pp. 208–230, 2011, pMID: 22149680. [Online]. Available: https://doi.org/10.3109/0954898X.2011.638355

[17] A. Krenker, J. Bešter, and A. Kos, "Introduction to the artificial neural networks," *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech*, pp. 1–18, 2011.

[18] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towards Data Sci*, vol. 6, no. 12, pp. 310–316, 2017.

[19] A. Abraham, "Artificial neural networks," *Handbook of measuring system design*, 2005.

[20] H. Tanaka, "Modeling the motor cortex: Optimality, recurrent neural networks, and spatial dynamics," *Neuroscience Research*, vol. 104, pp. 64–71, 2016, body representation in the brain. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168010215002631

[21] ——, "Modeling the motor cortex: Optimality, recurrent neural networks, and spatial dynamics," *Neuroscience Research*, vol. 104, pp. 64–71, 2016.

[22] L. Zhu, Q. Li, and R.-C. Lin, "A new approach for chaotic time series prediction using recurrent neural network," *Mathematical Problems in Engineering*, vol. 2016, p. 3542898, 2016.

[23] J. Zhang and K. Man, "Time series prediction using rnn in multi-dimension embedding phase space," in *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*, vol. 2, 1998, pp. 1868–1873 vol.2.

[24] H. V. Dudukcu, M. Taskiran, Z. G. Cam Taskiran, and T. Yildirim, "Temporal convolutional networks with rnn approach for chaotic time series prediction," *Applied Soft Computing*, vol. 133, p. 109945, 2023.

[25] M. Han, J. Xi, S. Xu, and F.-L. Yin, "Prediction of chaotic time series based on the recurrent predictor neural network," *IEEE Transactions on Signal Processing*, vol. 52, no. 12, pp. 3409–3416, 2004.

[26] E. Pekel and S. Kara, "A comprehensive review for artifical neural network application to public transportation," *Sigma Journal of Engineering and Natural Sciences*, vol. 35, pp. 157–179, 03 2017.

[27] P. Gu, R. Xiao, G. Pan, and H. Tang, "Stca: Spatio-temporal credit assignment with delayed feedback in deep spiking neural networks." in *IJCAI*, 2019, pp. 1366–1372.

[28] Y. H. Liu, A. Ghosh, B. Richards, E. Shea-Brown, and G. Lajoie, "Beyond accuracy: generalization properties of bio-plausible temporal credit assignment rules," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 23 077–23 097. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/9226f8122feb9c229c1efd9270ce7021-Paper-Conference.pdf

[29] T. P. Lillicrap and A. Santoro, "Backpropagation through time and the brain," *Current Opinion in Neurobiology*, vol. 55, pp. 82–89, 2019, machine Learning, Big Data, and Neuroscience. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959438818302009

[30] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *science*, vol. 304, no. 5667, pp. 78–80, 2004.

[31] A. Reprintsev and A. Reprintsev, "Turing completeness," *Oracle SQL Revealed: Executing Business Logic in the Database Engine*, pp. 235–242, 2018.

[32] A. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995.

[33] S. Siami-Namini, N. Tavakoli, and A. Siami Namin, "A comparison of arima and lstm in forecasting time series," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018, pp. 1394–1401.

[34] (2023, Mar) Module: Tf.keras :   tensorflow v2.12.0. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras

[35] (2022, May) Reservoirpy 0.3.6 documentation. [Online]. Available: https://reservoirpy.readthedocs.io/en/latest/index.html

[36] (2023, Apr) pandas.series.autocorr. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.Series.autocorr.html

[37] V. Kotu and B. Deshpande, "Chapter 4 - classification," in *Data Science (Second Edition)*, V. Kotu and B. Deshpande, Eds. Morgan Kaufmann, 2019, pp. 65–163.

[38] J. J. Berman, "Chapter 4 - understanding your data," in *Data Simplification*, J. J. Berman, Ed. Boston: Morgan Kaufmann, 2016, pp. 135–187.

[39] R. A. de Oliveira and M. H. Bollen, "Deep learning for power quality," *Electric Power Systems Research*, vol. 214, p. 108887, 2023.

[40] D. Singh and B. Singh, "Investigating the impact of data normalization on classification performance," *Applied Soft Computing*, vol. 97, p. 105524, 2020.

[41] A. Moberg. (2020, Jan) Stockholm historical weather observations - daily mean air temperatures since 1756: Bolin centre database. [Online]. Available: https://tinyurl.com/2nmcywz9

[42] R. G. Andrzejak, K. A. Schindler, and C. Rummel. (2012) Nonrandomness, nonlinear dependence, and nonstationarity of electroencephalographic recordings from epilepsy patients [dataset]. [Online]. Available: https://repositori.upf.edu/handle/10230/42829

[43] S. Wang, X. Yang, and Y. Li, "The mechanism of rotating waves in a ring of unidirectionally coupled lorenz systems," *Communications in Nonlinear Science and Numerical Simulation*, vol. 90, p. 105370, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1007570420302021