Model-Coupled Autoencoder for Time Series Visualisation

Nikolaos Gianniotis[a], Sven D. Kügler[a], Peter Tiňo[b], Kai L. Polsterer[a]

[a]*Astroinformatics Group, Heidelberg Institute for Theoretical Studies (HITS), Schloss-Wolfsbrunnenweg 35, 69118 Heidelberg, Germany*
[b]*School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, United Kingdom*

Abstract

We present an approach for the visualisation of a set of time series that combines an echo state network with an autoencoder. For each time series in the dataset we train an echo state network, using a common and fixed reservoir of hidden neurons, and use the optimised readout weights as the new representation. Dimensionality reduction is then performed via an autoencoder on the readout weight representations. The crux of the work is to equip the autoencoder with a loss function that correctly interprets the reconstructed readout weights by associating them with a reconstruction error measured in the data space of sequences. This essentially amounts to measuring the predictive performance that the reconstructed readout weights exhibit on their corresponding sequences when plugged back into the echo state network with the same fixed reservoir. We demonstrate that the proposed visualisation framework can deal both with real valued sequences as well as binary sequences. We derive magnification factors in order to analyse distance preservations and distortions in the visualisation space. The versatility and advantages of the proposed method are demonstrated on datasets of time series that originate from diverse domains.

Keywords: Time series, dimensionality reduction, echo state network, autoencoder

1. Introduction

Time series[1] are sequences of observations that exhibit short or long term dependencies between them in time. These dependencies can be thought of as manifestations of a latent regime (e.g. natural law) governing the behaviour of the time series. Machine learning approaches designed to deal with data of a vectorial nature have no knowledge of such temporal dependencies. By introducing a model that accounts for temporal behaviour, algorithms can become "aware" of the sequential nature of the data and make the most of the available information.

Echo state networks (ESNs) [1] are discrete time recurrent neural networks that have emerged as a popular method to capture the latent regime underlying a time series. ESNs have the great advantage that the hidden part, the reservoir of neurons, is fixed and only the output weights need to be trained. The ESN is thus essentially a linear model and so the output weights, also known as readout weights, can thus be easily optimised via least squares. The processing of structured data has been a topic of research for a long time [2, 3]. Regarding time series, recent attempts [4, 5, 6] have exploited the predictive capabilities of ESNs in regression and classification tasks. In the unsupervised setting, the work in [7] suggests compressing a linear state space model through a linear autoencoder in order to extract vectorial representations of structured data. The work in [8] considers the visualisation of individual observations belonging to a *single* sequence by temporally linking them using an ESN.

In this work, we employ the ESN in the formulation of a dimensionality reduction algorithm for visualising a dataset of time series (we extend previous work presented in [9]). Given a fixed reservoir, the only free parameters in the ESN are in the readout weight vector which maps the state space to the sequence space. Thus, an optimised (i.e. trained) readout weight vector uniquely addresses an instance of the ESN (always for the same fixed reservoir) that best predicts on a given sequence. We can also reason backwards: given an observed sequence, we can train the ESN (Section 2.1) and identify the readout weight vector that best predicts the given sequence. Hence, each sequence in the dataset can be mapped to the readout weight vector that exhibits the best predictive performance on it. These readout weight vectors in conjunction with the common and fixed reservoir, capture temporal features of their respective sequences. Representing sequences as weight vectors, constitutes the first part of our proposed approach (Section 3.1).

The second stage of our approach involves training an autoencoder [10] on the obtained readout weight vectors in order to induce a two-dimensional representation, the visualisation, at the bottleneck. At the heart of the autoencoder lies the reconstruction error function which drives the visualisation induced at the bottleneck. During training, the autoencoder receives readout weights as inputs, compresses them at the bottleneck, and outputs an approximate version of the inputs, the reconstructed readout weights. Typically, one would take as the reconstruction error function the L_2 norm between the original readout weights and reconstructed readout weights. In the proposed work, we equip the autoencoder with a different reconstruction function that assesses how well the reconstructed readout weights still predict on the sequence that it represents. If it predicts well, we deem it a good reconstruction; if it predicts poorly, we deem it a poor reconstruction (Section 3.2). An

[1]We interchangeably use the terms time series and sequence.

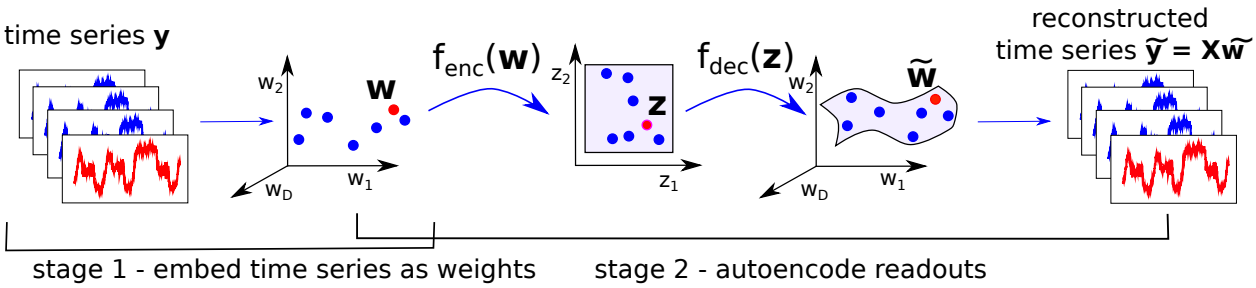


Fig. 1: Sketch of proposed method. In a first stage, time series $\boldsymbol{y}$ are cast to readout weights $\boldsymbol{w}$ in the weight space (see Section 3.1). In a second stage, the autoencoder projects readout weights $\boldsymbol{w}$ onto coordinates $\boldsymbol{z}$ residing in a two-dimensional space, and reconstructs them again as $\tilde{\boldsymbol{w}}$ (see Section 3.2). By multiplying with the state space, given by $\boldsymbol{X}$, we map the reconstructed readout weights $\tilde{\boldsymbol{w}}$ to the sequence space where reconstruction error is measured (see Eq. (7)).

overview of the proposed method is displayed in Fig. 1.

In Section 6, we show that the autoencoder with the proposed reconstruction error function is capable of interpreting similarities between time series better than other dimensionality reduction algorithms. In Section 7, we discuss the possibility of alternative formulations of the proposed approach before concluding with some remarks on future work in Section 8.

2. Preliminary

This section introduces some notation and terminology while briefly reviewing ESNs and the autoencoder.

2.1. Echo State Networks

An ESN is a discrete-time recurrent neural network with fading memory. It processes time series composed by a sequence of observations $y(t) \in \mathbb{R}$ over time t that we denote here by $\boldsymbol{y} = (y(1), \ldots, y(T))$, where T is the length[2] of the sequences. Hence $\boldsymbol{y} \in \mathbb{R}^{T\times 1}$. Given an input $y(t)$, the task of the ESN is to make a prediction $\hat{y}(t+1)$ for the observation $y(t+1)$ in the next time step. Similarly to a feedforward neural network, the ESN comprises an input layer with weights $\boldsymbol{v} \in \mathbb{R}^{D\times 1}$, a hidden layer with weights $\boldsymbol{U} \in \mathbb{R}^{D\times D}$ (hence D is the size of the reservoir) and an output layer with weights $\boldsymbol{w} \in \mathbb{R}^{D\times 1}$, the latter weights $\boldsymbol{w}$ also known as readout weights. However, in contrast to feedforward networks, ESNs equip the hidden neurons with feedback connections. The operation of an ESN is specified by the equations:

$$\boldsymbol{x}(t+1) = h(\boldsymbol{U}\boldsymbol{x}(t) + \boldsymbol{v}y(t)) \,, \tag{1}$$

$$\hat{y}(t+1) = \boldsymbol{w}^T \boldsymbol{x}(t+1) \,, \tag{2}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^{D\times 1}$ are the hidden activations of the reservoir at time t, and $h(\cdot)$ is a nonlinear function commonly chosen as the $\tanh(\cdot)$ function. Bias terms have been omitted in the formulation for the sake of clarity in notation.

According to standard ESN methodology [1], parameters $\boldsymbol{v}$ and $\boldsymbol{U}$ in Eqs. (1), (2) are randomly generated[3] and fixed. The only trainable parameters are the readout weights $\boldsymbol{w}$. Training involves feeding at each time step t an observation $y(t)$ and recording the resulting activations $\boldsymbol{x}(t)$ row-wise into a matrix $\boldsymbol{X} \in \mathbb{R}^{T\times D}$. Usually, some initial observations are dismissed in order to "washout" [1] dependence on the initial arbitrary reservoir state (e.g. $\boldsymbol{x}(1) = \boldsymbol{0}$). Given matrix $\boldsymbol{X}$, the following objective function is minimised:

$$\ell(\boldsymbol{w}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2 \,. \tag{3}$$

The above objective can be supplemented by a regularisation term and so the combined objective is $\ell(\boldsymbol{w}) + \mu^2\|\boldsymbol{w}\|^2$. The combined objective can be exactly minimised by solving the pertaining least squares problem and obtaining $\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X} + \mu^2\boldsymbol{I}_D)^{-1}\boldsymbol{X}^T\boldsymbol{y}$ as the solution, where $\boldsymbol{I}_D$ is the $D \times D$ identity matrix. Given this result, we introduce function $g(\boldsymbol{y})$ that maps a given time series to the optimal readout weights:

$$g(\boldsymbol{y}) = (\boldsymbol{X}^T\boldsymbol{X} + \mu^2\boldsymbol{I}_D)^{-1}\boldsymbol{X}^T\boldsymbol{y} = \boldsymbol{w} \,. \tag{4}$$

2.2. Deterministically Constructed Echo State Networks

In the original formulation of the ESN [1] the weights in $\boldsymbol{v}$ and $\boldsymbol{U}$ are generated stochastically and so are the connections between the hidden neurons in the reservoir. This makes the training and use of the ESN dependent on random initialisations. In order to avoid this source of randomness, we make use of a class of ESNs that are constructed in a deterministic fashion [11].

Deterministic ESNs make several simplifications over standard ESNs. All entries in $\boldsymbol{v}$ have the same absolute value of a single scalar parameter $v > 0$. The signs of the entries in $\boldsymbol{v}$ are deterministically generated by an aperiodic sequence: e.g. a pseudorandom binary sequence (coin flips), with outcomes 0 and 1 corresponding to $-$ and $+$ respectively. Similarly, the entries in $\boldsymbol{U}$ are parametrised by a single scalar $u > 0$. As opposed to random connectivity, deterministic ESNs impose a

[2] In general, each sequence can have its own length T_n. For ease of exposition, here all sequences have the same T.

[3] The spectral radius of the reservoir's weight matrix $\boldsymbol{U}$ is made < 1 to encourage *the Echo State Property*.

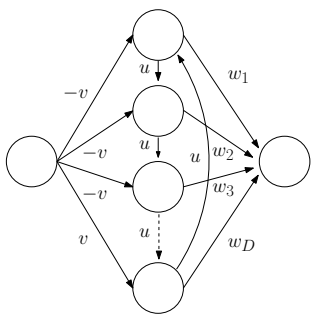


Fig. 2: Deterministic ESN with cyclic architecture, see Section 2.2. Circles denote neurons and arrows connections between neurons. All input neurons connect to the hidden neurons, and all hidden neurons connect to the output neurons. Hidden neurons are connected in a cyclic fashion to each other. All input weights have the same absolute value v, and the sign is determined by a deterministic aperiodic sequence. The hidden reservoir weights are fixed to the same scalar u. The readout weights $\boldsymbol{w}$ are the only adaptable part of the ESN.

fixed regular topology on the hidden neurons in the reservoir. Amongst possible choices, one can arrange the neurons in a cycle. A cyclic arrangement imposes the following structure on $\boldsymbol{U}$: the only nonzero entries in $\boldsymbol{U}$ are on the lower sub-diagonal $\boldsymbol{U}_{i+1,i} = u$, and at the upper-right corner $\boldsymbol{U}_{1,D} = u$. An illustration of a cyclic deterministic ESN is shown in Fig. 2.

In this work we employ deterministic ESNs with a cyclic connectivity. Deterministic ESNs have three degrees of freedom: the reservoir size D, the input weight v and reservoir weight u. Hence, the triple (D, v, u) completely specifies an ESN. It has been shown empirically and theoretically (memory capacity) [11] that deterministic ESNs perform up to par with their stochastic counterparts. Training of a deterministic ESN is performed in exactly the same fashion as in stochastically constructed ESNs using the objective $\ell(\boldsymbol{w})$ in Eq. (3).

2.3. Autoencoder

The autoencoder [10] is a feedforward neural network that defines a three hidden layer architecture[4] with the middle layer, the "bottleneck", being smaller than the others in terms of the number of neurons denoted by Q. The autoencoder learns an identity mapping by training on targets identical to the inputs. Learning is hampered by the bottleneck that forces the autoencoder to reduce the dimensionality of the inputs, and hence the output is only an approximate reconstruction of the input.

Given general vectors $\boldsymbol{s}$, we want to reduce them to a Q-dimensional representation. The autoencoder is the composition of an encoding f_{enc} and a decoding f_{dec} function. Encoding maps inputs $\boldsymbol{s}$ to low-dimensional compressed versions, $f_{enc}(\boldsymbol{s}) = \boldsymbol{z} \in \mathbb{R}^Q$, while decoding maps approximately back to the inputs, $f_{dec}(\boldsymbol{z}) = \tilde{\boldsymbol{s}}$. The complete autoencoder is the function $f(\boldsymbol{s}; \boldsymbol{\theta}) = f_{dec}(f_{enc}(\boldsymbol{s})) = \tilde{\boldsymbol{s}}$, where $\boldsymbol{\theta}$ are the weights of the autoencoder. Training the autoencoder involves minimising the L_2 norm between N given vectors $\boldsymbol{s}$ and their reconstructions:

$$\sum_{n=1}^{N} \|\tilde{\boldsymbol{s}}_n - \boldsymbol{s}_n\|^2 = \sum_{n=1}^{N} \|f(\boldsymbol{s}_n; \boldsymbol{\theta}) - \boldsymbol{s}_n\|^2. \tag{5}$$

3. Model Formulation

The proposed approach consists of two stages. In Section 3.1, we discuss how time series $\boldsymbol{y}$ are embedded in the space of readout weight vectors $\boldsymbol{w}$. Section 3.2 discusses how an autoencoder with a modified reconstruction function is applied on the readout weight vectors in a meaningful manner.

3.1. Embedding time series in the space of readout weights

Given a deterministically constructed and fixed reservoir (D, v, u), we cast a dataset $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\}$ via $g(\boldsymbol{y}_n) = \boldsymbol{w}_n$ to a new dataset of readout weights $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N\}$. *We emphasise that all time series are embedded in the space of readout weights with respect to the same fixed dynamic reservoir* (D, v, u). After this embedding, visualisation proceeds by performing dimensionality reduction on the new representations $\boldsymbol{w}_n$. We take the view that the readout weight $\boldsymbol{w}_n$ is a good representation for a sequence $\boldsymbol{y}_n$ with respect to the fixed reservoir (D, v, u). The readout weight $\boldsymbol{w}_n$ captures important information about $\boldsymbol{y}_n$ in the sense that it exhibits good predictive power on it. Moreover, the readout weight vector $\boldsymbol{w}_n$ features time-shift invariance, and can accommodate sequences of variable length.

A prerequisite for a successful embedding is a common, fixed reservoir that enables good predictive performance on the data. To find this reservoir, we opt for a simple strategy. For both v and u we take a regular grid of e.g. 10 candidate values $[10^{-2}, \ldots, 1.0]$. For each combination $(u, v) \in [10^{-2}, \ldots, 1.0] \times [10^{-2}, \ldots, 1.0]$, we perform the following:

1. Split each sequence $\boldsymbol{y}$ in two halves $\boldsymbol{y}_n^{(train)}$ and $\boldsymbol{y}_n^{(test)}$.
2. According to Eq. (3), train ESN on $\boldsymbol{y}_n^{(train)}$ by minimising $\ell^{(train)}(\boldsymbol{w}) = \|\boldsymbol{X}_n^{(train)}\boldsymbol{w} - \boldsymbol{y}_n^{(train)}\|^2$ which yields $\boldsymbol{w}_n$.
3. Measure test error via $\ell^{(test)}(\boldsymbol{w}_n) = \|\boldsymbol{X}_n^{(test)}\boldsymbol{w}_n - \boldsymbol{y}_n^{(test)}\|^2$.

Matrices $\boldsymbol{X}_n^{(train)}$ and $\boldsymbol{X}_n^{(test)}$ respectively record row-wise the activations $y_n^{(train)}(t)$ and $y_n^{(test)}(t)$ as specified in Section 2.1. The combination (u, v) with the lowest test error over all sequences $\sum_{n=1}^{N} \ell^{(test)}(\boldsymbol{w}_n)$, determines the ESN that will cast all time series in the dataset to readout weights. Parameters D and μ may also be included in this simple validation scheme.

3.2. ESN-coupled Autoencoder

We want to reduce the dimensionality of the new representations $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N\}$ using an autoencoder. One possibility is to directly apply the autoencoder taking as input the readout weights and returning their reconstructed versions, $f : \mathbb{R}^{D\times 1} \to \mathbb{R}^{D\times 1}$. We could then minimise the following objective function with respect to the autoencoder weights $\boldsymbol{\theta}$:

$$\sum_{n=1}^{N} \|f(\boldsymbol{w}_n; \boldsymbol{\theta}) - \boldsymbol{w}_n\|^2 = \sum_{n=1}^{N} \|\tilde{\boldsymbol{w}}_n - \boldsymbol{w}_n\|^2. \tag{6}$$

[4] To be perfectly precise, we use what is widely considered the standard autoencoder specified in [12, Sec. 12.4.2]).

A limitation of the above objective function is that it merely measures how well the reconstructions $f(\boldsymbol{w};\boldsymbol{\theta}) = \tilde{\boldsymbol{w}}$ approximate the original inputs $\boldsymbol{w}$ in the L_2 sense.

A better objective would measure reconstruction error in the sequence space as opposed to the space of readout weights. To that purpose, we map the reconstructed readout weights $\tilde{\boldsymbol{w}}$ to the sequence space by multiplying with the respective state matrix, $\boldsymbol{X}\tilde{\boldsymbol{w}} = \tilde{\boldsymbol{y}}$. In actual fact, function $\ell(\cdot)$ in Eq. (3) is cut out for this task: if $\ell(\tilde{\boldsymbol{w}})$ returns high likelihood, then $\tilde{\boldsymbol{w}}$ is a good reconstruction; if $\ell(\tilde{\boldsymbol{w}})$ returns low likelihood, then $\tilde{\boldsymbol{w}}$ is a poor reconstruction. The new objective function reads:

$$\sum_{n=1}^{N} \ell_n(f(\boldsymbol{w}_n;\boldsymbol{\theta})) = \sum_{n=1}^{N} \|\boldsymbol{X}_n f(\boldsymbol{w}_n;\boldsymbol{\theta}) - \boldsymbol{y}_n\|^2 = \sum_{n=1}^{N} \|\tilde{\boldsymbol{y}}_n - \boldsymbol{y}_n\|^2 \,, \quad (7)$$

where ℓ_n and $\boldsymbol{X}_n$ are respectively the objective function and state space pertaining to sequence $\boldsymbol{y}_n$, see Eq. (3). The gradient of the new objective function in Eq. (7) with respect to the weights $\boldsymbol{\theta}$, is calculated by backpropagation [12]. We use L-BFGS as the optimisation routine for training the weights $\boldsymbol{\theta}$.

3.3. Data Projection

Having trained the autoencoder $f(\boldsymbol{w}_n;\boldsymbol{\theta})$, we would like to project a time series $\boldsymbol{y}^*$ to a coordinate $\boldsymbol{z}^* \in \mathbb{R}^Q$. To that end, we first use the fixed ESN reservoir to cast the time series to $g(\boldsymbol{y}^*) = \boldsymbol{w}^*$. Then, the readout weight $\boldsymbol{w}^*$ is projected using the encoding part of the autoencoder to obtain $f_{enc}(\boldsymbol{w}^*) = \boldsymbol{z}^*$.

4. Binary Sequences

The time series considered so far are sequences of reals $y(t) \in \mathbb{R}$. However, it is possible to extend the proposed approach to the processing of symbolic sequences. In particular, we consider binary sequences composed of observations $y(t) \in \{0, 1\}$. For an ESN to process binary sequences, we pass its outputs through the logistic function $\sigma(\cdot) = (1 + \exp(\cdot))^{-1}$ (link function of the Bernoulli distribution). Hence, the equations specifying the operation of the ESN now read[5]:

$$\boldsymbol{x}(t+1) = h(\boldsymbol{U}\boldsymbol{x}(t) + \boldsymbol{v}(y(t) - 0.5)) \,, \quad (8)$$

$$\hat{y}(t+1) = \sigma(\boldsymbol{w}^T \boldsymbol{x}(t+1)) \,. \quad (9)$$

Here the output $\hat{y}(t+1) \in [0 \ldots 1]$ of the ESN is interpreted as the probability of the next observation $y(t+1)$ being equal to 1, i.e. $\hat{y}(t+1) = p(y(t+1) = 1)$. Accordingly, the objective function $\ell(\boldsymbol{w})$ in Eq. (3) needs to be redefined. Instead of solving a least squares problem, we minimise the cross-entropy:

$$\ell^{ce}(\boldsymbol{w}) = -\sum_{t=1}^{T} y(t) \log \hat{y}(t) \,. \quad (10)$$

Training of the ESN is carried out by iterative gradient minimisation of Eq. (10) preceded by a period of washout.

[5] In Eq. (8) we subtract 0.5 from $y(t)$, since the symmetric $\tanh(\cdot)$ transfer function h is used in the dynamic reservoir.

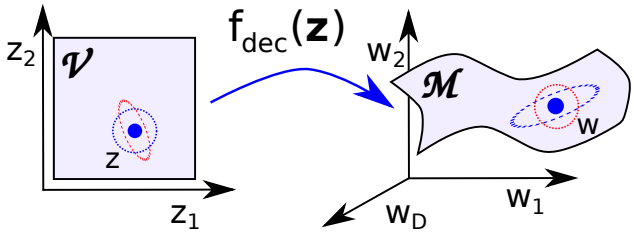


Fig. 3: Stylised sketch: mapping $f_{dec}(\boldsymbol{z})$ embeds the visualisation space $\mathcal{V}$ as a manifold $\mathcal{M}$ in the space of readout weights. Each point $\boldsymbol{z}$ addresses a probabilistic ESN with readout weights $\boldsymbol{w}$.

The above modifications to the ESN, call for a modification also in the autoencoder. While in Eq. (7) reconstruction is measured via the least-squares based function $\ell(\boldsymbol{w})$, we now use the cross-entropy based function $\ell^{ce}(\boldsymbol{w})$. In order for the autoencoder to process correctly the weights coming from binary sequences, its objective function needs to be changed to:

$$\sum_{n=1}^{N} \ell_n^{ce}(f(\boldsymbol{w}_n;\boldsymbol{\theta})) = -\sum_{n=1}^{N}\sum_{t=1}^{T} y(t) \log \sigma(f(\boldsymbol{w}_n;\boldsymbol{\theta})^T \boldsymbol{x}_n(t+1)) \,. \quad (11)$$

In the case of binary sequences, the outputs of the autoencoder $f(\boldsymbol{w};\boldsymbol{\theta})$ are put though the function $\ell^{ce}(\cdot)$.

By adopting a 1-of-K coding scheme for the symbols, and the softmax function in the place of the logistic function, an extension to K number of symbols is possible. The resulting objective for training the ESN is again a cross-entropy function.

5. Magnification Factors

In Fig. 3, the smooth nonlinear function $f_{dec}(\boldsymbol{z})$ embeds the low-dimensional visualisation space $\mathcal{V}$ as a Q-dimensional manifold $\mathcal{M}$ in the space of readout weights $\boldsymbol{w}$. Each point $\boldsymbol{z} \in \mathcal{V}$ addresses an ESN model[6] with readout weights $\boldsymbol{w} \in \mathcal{M}$. The ESN model may be viewed as a probabilistic model, if we assume that observations $y(t)$ are corrupted by i.i.d. Gaussian noise of zero mean and variance ϵ^2:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + \epsilon \,, \quad (12)$$

$$p(\boldsymbol{y};\boldsymbol{w}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{X}\boldsymbol{w}, \epsilon \boldsymbol{I}_T) \,, \quad (13)$$

Thus, each point $\boldsymbol{z}$ addresses a probabilistic model $p(\boldsymbol{y}; f_{dec}(\boldsymbol{z}))$, and $\mathcal{M}$ is a manifold of probabilistic models $p(\boldsymbol{y}; f_{dec}(\boldsymbol{z}))$.

Manifold $\mathcal{M}$ is endowed with a natural metric for measuring distances between probabilistic models $p(\boldsymbol{y}; f_{dec}(\boldsymbol{z}))$. Specifically, the metric tensor on a statistical manifold at a given point $\boldsymbol{z}$ is the $Q \times Q$ Fisher information matrix (FIM) [13]. Here, we approximate it through the *observed FIM* over the given dataset of sequences:

$$\boldsymbol{F}(\boldsymbol{z})_{i,j} = -\sum_{n=1}^{N} \left[\left(\frac{\partial}{\partial z_i} \log p(\boldsymbol{y}_n; f_{dec}(\boldsymbol{z})) \right) \left(\frac{\partial}{\partial z_j} \log p(\boldsymbol{y}_n; f_{dec}(\boldsymbol{z})) \right) \right] . \quad (14)$$

[6] We always have the same fixed reservoir.

We note that the visualisation space $\mathcal{V}$ does not necessarily reflect distances between models on $\mathcal{M}$. In Fig. 3, we see how neighbourhoods of $\boldsymbol{z}$, depicted as dotted ellipses, transform on $\mathcal{M}$. Thus, in order to interpret distances in $\mathcal{V}$, it is important to push-forward the natural notion of distances on $\mathcal{M}$ onto the visualization space $\mathcal{V}$. In the topographic mapping literature the induced metric in the visualization space from the data space is usually represented through magnification factors [14]. In the following we show how magnification factors can be computed in the ESN-AE setting.

Given the FIM, one can push forward local distances $\Delta\boldsymbol{z}$ from $\mathcal{M}$ onto $\mathcal{V}$ via $\Delta\boldsymbol{z}^T\boldsymbol{F}(\boldsymbol{z})\Delta\boldsymbol{z}$. In particular, at a given point $\Delta\boldsymbol{z}$ it is possible to estimate in which direction $d\boldsymbol{z}$ the distance changes the most. This can be easily calculated by solving the following constrained problem:

$$\text{maximise } \Delta\boldsymbol{z}^T\boldsymbol{F}\Delta\boldsymbol{z} \quad \text{over } \Delta\boldsymbol{z}, \quad \text{subject to } \|\Delta\boldsymbol{z}\|^2 = 1. \tag{15}$$

The solution to this problem is given by setting $\Delta\boldsymbol{z}^*$ to the eigenvector corresponding to the largest eigenvalue λ^*. Eigenvalue λ^* informs us of the maximum local distortion in distance and can be taken as a measure for the local magnification factor.

6. Numerical Experiments

In the following we compare the proposed method to other visualisation algorithms and discuss the results.

6.1. Datasets

In order to judge whether a visualisation truly captures similarities, we need to know a priori which time series are similar to which. We therefore employ the following particular datasets whose data items fall under known classes and are labelled. For these datasets, there is a very strong a priori expectation that the classes *are governed by qualitatively distinct dynamical regimes*. Thus, time series of the same class are expected to appear similar (close together) in the visualisation, while time series belonging to different classes are expected to appear dissimilar (separate) in the visualisation.

NARMA. We generate 100 sequences of length 1000 from the three qualitatively different NARMA classes [11] of orders $10, 20, 30$. The NARMA time series is an interesting benchmark problem due to the presence of long-term dependencies.

Cauchy. We sample sequences from a stationary Gaussian process with correlation function given by $c(x_t, x_{t+h}) = (1 + |h|^a)^{-\frac{a}{b}}$ [15]. We generated 4 classes by permuting parameters $a \in \{0.65, 1.95\}$ and $b \in \{0.1, 0.95\}$. We generated from each class 100 time series of length $1,000$. Parameters a and b are respectively related to the fractal dimension (measuring self-similarity) and the Hurst coefficient (measuring long-memory dependence) of the series. By construction, the four classes have distinct characteristics.

X-ray. The binary system GRS1915+105 is composed of an extremely heavy stellar black hole and a low-mass star. Material is transferred from the star towards the black hole through the Roche lobe. While falling into the gravitational potential of the black hole, energy is released by radiating X-ray and radio (jet) emission which is typical for the class of microquasars. A thorough investigation carried out in [16], detected the presence of classes of distinct dynamical patterns. Due to the lack of multiple time sequences per state, we split the observations into equal-length parts, resulting in 161 sequences. Here we visualise classes *delta*, *kappa*, *phi*, *rho* and *chi*.

Wind. We visualise wind data[7] taken from the vicinity of Hamburg, Frankfurt and Munich. Around each city, we select the 10 closest stations with a completeness of more than 99% of hourly measured wind speed data between 13/01/2014 - 31/12/2014 ($8,471$ measurements per station). Missing data are interpolated using a spline function of the $3rd$ degree. In order to increase the number of visualised entities, the time series of each station are cut into two non-overlapping parts of $4,000$ data points each. In these data there is a strong a priori expectation that time series associated with the coastal city of Hamburg are different to the other data.

Textual data (symbolic). We visualise the first chapter of J. K. Rowling's "Harry Potter and the Philosopher's Stone" in three languages German, English and Spanish. A full symbolic representation of the alphabet makes the optimisation of the ESN difficult and it would be a trivial task to separate the languages as they could be identified by single words. Here, we choose a binary representation where the states 0 and 1 represent vowels and consonants. Punctuation and whitespaces are ignored. E.g. a German sentence is converted as follows:

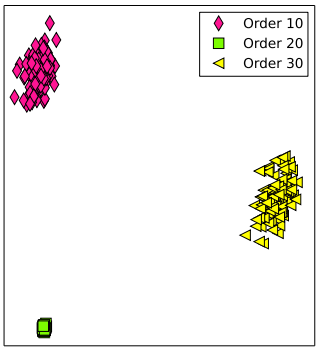
≫Die Potters, das stimmt, das hab ich gehört -≪
_011_0100100__010_001000__010_010_100_010100___

Discarded symbols are marked by an underscore. This representation returns sequences of different length for each language, but all with at least $24,000$ symbols. To increase the number of sequences per language, we split the binary vectors into sequences of length $2,000$ with neighbouring sequences overlapping by 50%. It is interesting to see whether texts originating from different languages still retain their distinguishing dynamics after subjected to this drastic "binarisation".

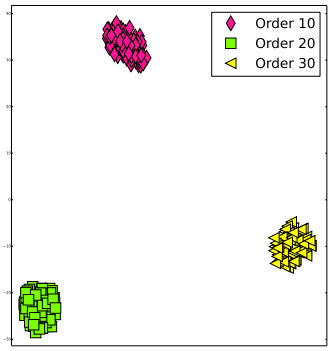
6.2. Dimensionality Reduction Algorithms

The following dimensionality reduction algorithms are compared in the numerical experiments. All algorithms operate on the readout weights $\boldsymbol{w}$. Sequences are represented as readout weights using a deterministic cyclic ESN whose parameters are selected using the validation procedure in Section 3.1. Additionally, in this validation scheme we include the regularisation parameter $\mu \in \{10^{-2}, 10^{-3}, 10^{-4}\}$. In all experiments the size of the reservoir is fixed to $D = 50$ and we set a washout period of 50 time steps. We set $Q = 2$ for constructing 2D visualisations.

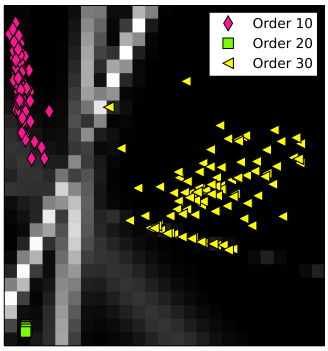
[7] Kindly provided by the Deutscher Wetterdienst, ftp://ftp-cdc.dwd.de/ .



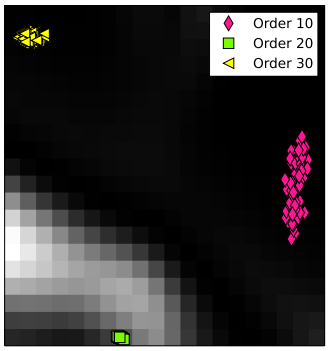
(a) PCA on NARMA.



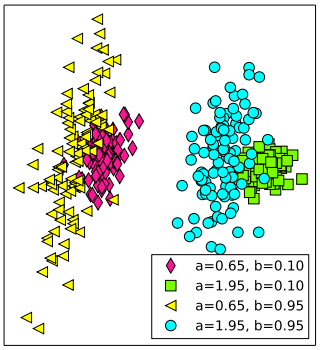
(b) t-SNE on NARMA, perpl.=40.



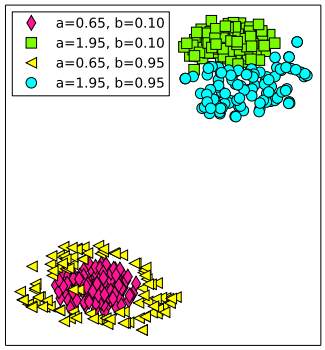
(c) Standard-AE on NARMA.



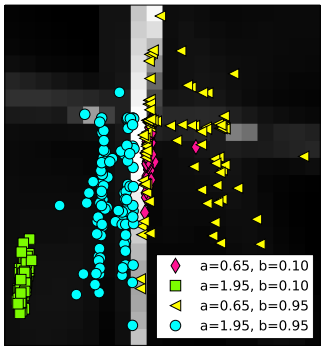
(d) ESN-AE on NARMA.



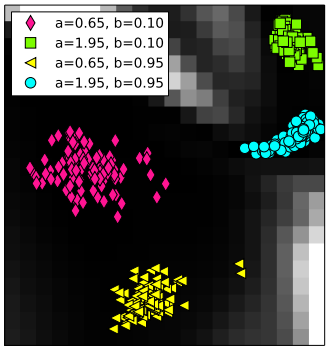
(e) PCA on Cauchy.



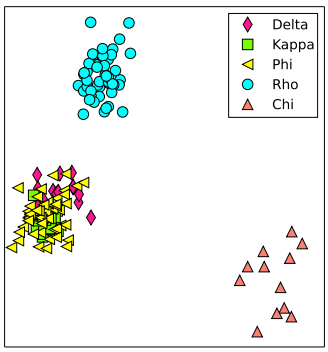
(f) t-SNE on Cauchy, perpl.=30.



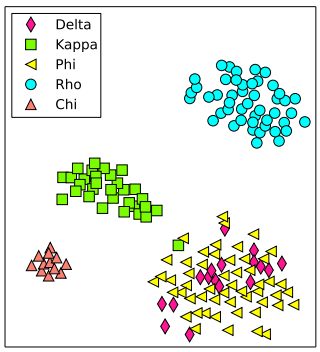
(g) Standard-AE on Cauchy.



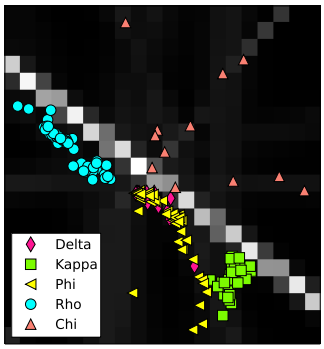
(h) ESN-AE on Cauchy.



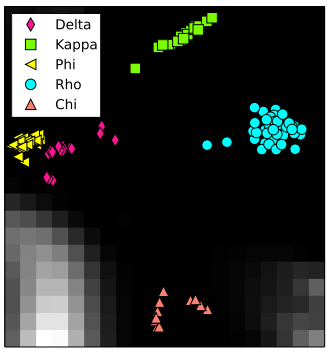
(i) PCA on X-ray.



(j) t-SNE on X-ray, perpl.=10.



(k) Standard-AE on X-ray.



(l) ESN-AE on X-ray.

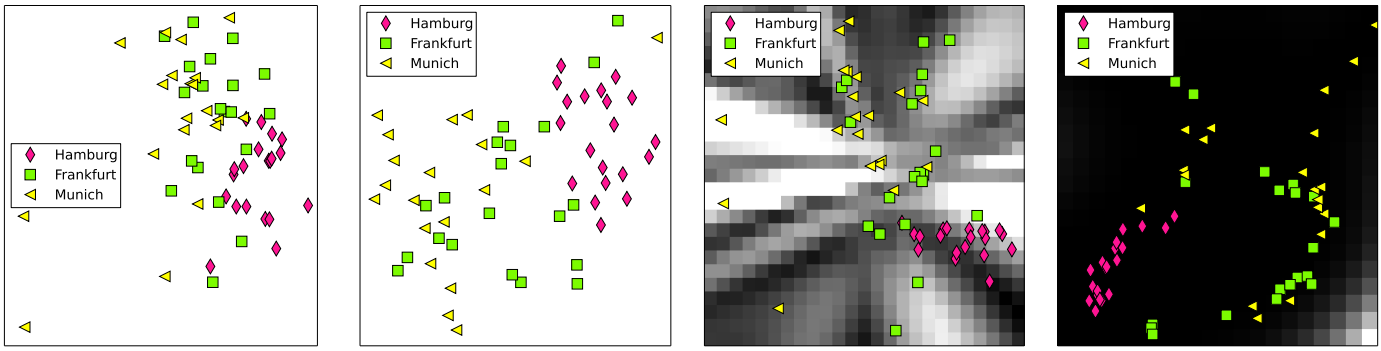
Fig. 4: Visualisations on NARMA (top), Cauchy (middle) and X-ray (bottom) data. High/low magnifications correspond to bright/dark regions. Legends specify which markers correspond to which classes.

PCA. We include PCA as it helps us gauge how difficult it is to project a dataset to low dimensions: if PCA delivers a good result, this hints that a complex, non-linear projection is superfluous.

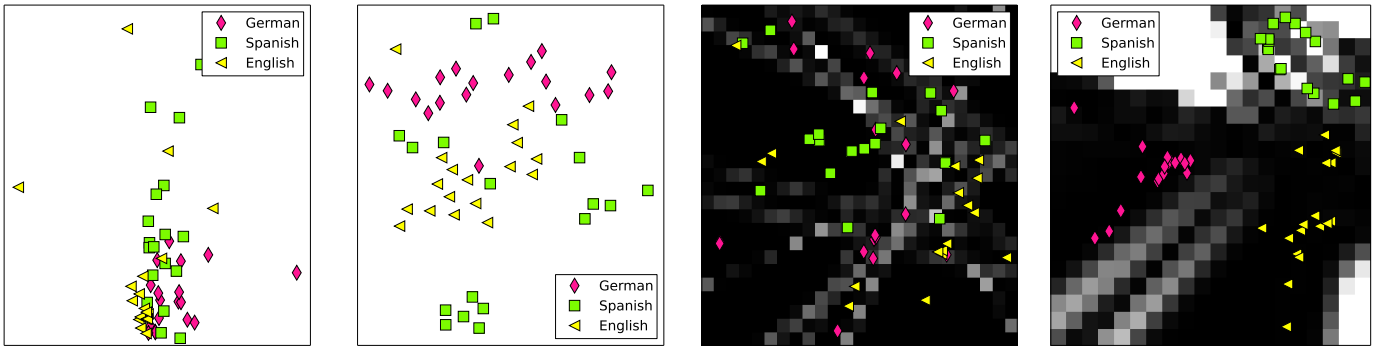
t-SNE. We include t-SNE [17] as one of the most popular and well performing algorithms designed for vectorial data. We train t-SNE with perplexities in $[5, 10, 20, 30, 40, 50]$, and display the visualisation that shows the best class separation. The chosen perplexity is quoted in the figures.

Standard autoencoder (standard-AE). We employ the standard autoencoder operating directly on the readout weights. The hidden layers of the encoding and decoding part have the same number H of neurons. We also add a regulariser on the weights of the autoencoder $\nu^2\|\boldsymbol{\theta}\|^2$ to control complexity. In all experiments, we set $H = 10$, $\nu = 1$.

Proposed approach (ESN-AE). The proposed ESN-AE has the same hyperparameters as the standard-AE. We again fix the hyperparameters to $H = 10$, $\nu = 1$.



(a) PCA on Wind. (b) t-SNE on Wind, perpl.=5. (c) Standard-AE on Wind. (d) ESN-AE on Wind.



(e) PCA on Textual. (f) t-SNE on Textual, perpl.=10. (g) Standard-AE on Textual. (h) ESN-AE on Textual.

Fig. 5: Visualisations on Wind (top), and Textual (bottom) data. High/low magnifications correspond to bright/dark regions. Legends specify which markers correspond to which classes.

6.3. Results

We present the visualisations in Figs. 4 and 5. Each column of plots corresponds to one of the aforementioned dimensionality reduction algorithms, and each row to a dataset. The projections in the plots appear as coloured markers of different shapes indicating class origin. The legend in each plot shows which marker corresponds to which class. Following Section 5, we display local magnification factors, for the autoencoders, as the maximum eigenvalue λ^* of matrix $\boldsymbol{F}(\boldsymbol{z})$ on a regular grid of points $\boldsymbol{z}$ on the visualisation space. Dark and bright values signify low and high eigenvalues/magnification factors respectively. There are no magnification factors for PCA, as the linear mapping connecting the visualisation space to the high-dimensional space is length/distance preserving. Also, we do not present magnification factors for t-SNE, as it does not define an explicit mapping between the visualisation and high-dimensional space. It thus requires a different framework than the one used here in order to study magnifications.

NARMA, *top row in Fig. 4.* We note that all visualisations separate the three classes, and show that the three classes are equidistant. The magnifications in 4c show that the standard-AE views the three classes indeed as distinctly separable clusters. However, in the case of the ESN-AE in 4d, the magnification factors suggest the presence of distortions in distances close to class "Order 20". This means that in actual fact class "Order 20" is separated by significant distance from the other two classes, and that classes "Order 10" and "Order 30" are closer and more similar to each other. We investigate this hypothesis with a simple experiment. We generate from each class additional 200 sequences. For each pair of classes (classes also pair with themselves), we train on sequences from one class and measure the mean squared error on the unseen sequences of the other classes. These errors are reported in Table 1, and support this hypothesis put forward by the magnification factors in the ESN-AE visualisation.

Table 1: Mean squared errors between NARMA classes, the smaller the more similar.

	Order 10	Order 20	Order 30
Order 10	5.331	2185.935	37.161
Order 20	2213.019	0.052	2030.409
Order 30	30.478	1983.585	6.031

Cauchy, *middle row in Fig. 4.* PCA in 4e and t-SNE in 4f manage to organise the classes coherently to some degree, while the standard-AE in 4g fails to produce a convincing result. ESN-AE in 4h displays a clear separation between all four classes.

Table 2: Mean reconstruction and standard deviation, averaged over 10 runs.

	PCA	standard AE	ESN-AE
NARMA	151451.130 ± 14984.801	116.041 ± 46.606	44.126 ± 11.962
Cauchy	121.110 ± 4.022	102.115 ± 2.522	95.176 ± 2.675
Xray	25.376 ± 2.274	42.727 ± 17.423	21.798 ± 1.617
Wind	5.0498 ± 0.253	5.192 ± 0.260	5.079 ± 0.231
Textual	0.691 ± 0.007	0.700 ± 0.010	0.694 ± 0.017

In particular the presence of magnification factors close to the two classes located in the upper right corner, shows that these two classes are potentially separated by a larger distance to the other two.

X-ray, *bottom row in Fig. 4.* All visualisations clearly separate the *rho* and *chi* classes. For standard-AE in 4k, the strong magnification suggest that the *chi* class is quite different to the others. t-SNE in 4j and ESN-AE distinguish in 4l the classes in a clearer fashion. ESN-AE exhibits less overlapping projections, but does not put enough distance between classes *delta* and *phi*. The presence of magnifications close to the *chi* class is a hint that this class is quite different to the other ones. Still even in the absence of labels (i.e. colour markers), the classes are identifiable in the visualisation produced by ESN-AE.

Wind, *top row in Fig. 5.* None of the visualisations separates the Munich from the Frankfurt stations. Matching our prior expectation, ESN-AE in 5d organises the stations around Hamburg in a single region, in contrast to the other visualisations which show overlap. Standard-AE fails to produce a clear result and its magnifications do not help in its interpretation any further.

Textual data (symbolic), *bottom row in Fig. 5.* The binary representation of the text data in three different languages shows the true power behind the ESN-AE equipped here with the logistic function. While other visualisations do not exhibit adequate separation, ESN-AE in 5h exhibits some clear organisation. Additionally, magnifications suggest some separation between the German and English sequences. The bright magnifications that appear in the unpopulated corners are simply artefacts as the model has not seen any data in these areas.

Reconstruction. In order to give a quantitative impression of the quality of the visualisations, we report reconstruction errors in Table 2. Each dataset is randomly partitioned 10 times into equally sized training and test sets. For each partitioning, we train the dimensionality algorithms and measure the error on the test data using Eq. (3). For the binary textual data, the error is measured as the fraction of predictions coinciding with the binary test sequences. We exclude t-SNE as it does not offer a way of reconstructing weights from the projections.

7. Discussion

Though the conversion of time-series into fixed-length representations is not new (e.g. [18]), we believe that converting the time series via a non-parametric state space model with fixed dynamic part (i.e ESN) in conjunction with an appropriately defined reconstruction function, does provide a new way of performing dimensionality reduction on time series. The results show that the proposed visualisation is better at understanding what makes sequences (dis)similar as it manages to separate classes that are governed by qualitatively distinct dynamical regimes. Indeed, the produced visualisations reflect our prior expectations as to which sequences should be similar.

Of course, combining the ESN with the autoencoder is just one possible scheme, and certainly other dimensionality reduction schemes can be devised along this line. One can exchange the ESN with other models such as autoregressive-moving-average models (ARMA), and use them to cast the time-series to fixed parameter vectors. E.g. for slow changing signals, models based on Fourier series might be more suitable than the ESN. Choosing the ESN to model the temporal features of the sequences, is indeed a subjective choice. However, this does not mean that it is a bad choice: in the relevant literature, a wealth of applications demonstrate that ESNs are good models for a large variety of real-world time series.

Besides the autoencoder, other dimensionality reduction methods that rely on optimising reconstruction error (e.g. GPLVM [19]) can be adapted to the visualisation of time-series; one has to modify their objective to measure reconstruction in the sequence space, just as the loss function of ESN-AE does.

8. Conclusion

We have presented a method for the visualisation of time series that couples an ESN to an autoencoder. Time series are represented as readout weights of an ESN and are subsequently compressed to a low dimensional representation by an autoencoder. The autoencoder attempts reconstruction of the readout weights in the context of the state space pertaining to the sequences thanks to the modified loss function. In future research, we plan to work on irregularly sampled time series that originate from eclipsing binary stars. The ESN will be replaced by a physical model that will cast the time series to vectors of physical parameters.

Acknowledgement

The authors from HITS gratefully acknowledge the support of the Klaus Tschira Foundation. We thank Ranjeev Misra for providing the X-ray dataset. We would also like to thank the anonymous reviewers for helping us improve the presented work.

[1] M. Lukosevicius, H. Jaeger, Reservoir computing approaches to recurrent neural network training, Computer Science Review 3 (3) (2009) 127–149.

[2] T. Jaakkola, D. Haussler, Exploiting generative models in discriminative classifiers, in: NIPS, 1998, pp. 487–493.

[3] T. Jebara, R. Kondor, A. Howard, Probability product kernels, Journal of Machine Learning Research 5 (2004) 819–844.

[4] S. Chatzis, Y. Demiris, Echo state gaussian process, Neural Networks, IEEE Transactions on 22 (9) (2011) 1435–1445.

[5] H. Chen, F. Tang, P. Tino, X. Yao, Model-based kernel for efficient time series analysis, in: KDD, 2013, pp. 392–400.

[6] H. Chen, P. Tino, A. Rodan, X. Yao, Learning in the model space for cognitive fault diagnosis, IEEE TNNLS 25 (1) (2014) 124–136.
[7] A. Sperduti, Linear autoencoder networks for structured data, in: International Workshop on Neural-Symbolic Learning and Reasoning, 2013.
[8] T.-D. Wang, X. Wu, C. Fyfe, Comparative study of visualisation methods for temporal data, in: IEEE CEC, 2012, pp. 1052–1056.
[9] N. Gianniotis, S. Kuegler, R. M. P. Tino, K. Polsterer, Autoencoding time series for visualisation, in: ESANN, 2015.
[10] M. A. Kramer, Nonlinear principal component analysis using autoassociative neural networks, AICHE Journal 37 (1991) 233–243.
[11] A. Rodan, P. Tino, Minimum complexity echo state network, IEEE Transactions on Neural Networks 22 (1) (2011) 131–144.
[12] C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[13] S. Kullback, Information theory and statistics, Wiley, New York, 1959.
[14] C. Bishop, M. Svensen, C. K. I. Williams, Magnification factors for the GTM algorithm, in: ICANN, 1997, pp. 64–69.
[15] T. Gneiting, M. Schlather, Stochastic models that separate fractal dimension and the hurst effect, SIAM Review 46 (2) (2004) 269–282.
[16] T. Belloni, M. Klein-Wolt, M. Méndez, M. van der Klis, J. van Paradijs, A model-independent analysis of the variability of GRS 1915+105, Astronomy & Astrophysics 355 (2000) 271–290.
[17] L. van der Maaten, G. Hinton, Visualizing data using t-SNE, JMLR 9 (2008) 2579–2605.
[18] J. Grabocka, M. Wistuba, L. Schmidt-Thieme, Scalable classification of repetitive time series through frequencies of local polynomials, IEEE TKDE 27 (6) (2015) 1683–1695.
[19] N. D. Lawrence, Probabilistic non-linear principal component analysis with gaussian process latent variable models, JMLR 6 (2005) 1783–1816.