A fast noise filtering algorithm for time series prediction using recurrent neural networks

Boris Rubinstein,
Stowers Institute for Medical Research
1000 50th St., Kansas City, MO 64110, U.S.A.

September 23, 2020

Abstract

Recent researches demonstrate that prediction of time series by recurrent neural networks (RNNs) based on the noisy input generates a *smooth* anticipated trajectory. We examine the internal dynamics of RNNs and establish a set of conditions required for such behavior. Based on this analysis we propose a new approximate algorithm and show that it significantly speeds up the predictive process without loss of accuracy and demonstrates increased robustness in neuroscience context.

1 Introduction

Recurrent neural networks (RNNs) due to their ability to process sequences of data have found applications in many fields of science, engineering and humanities, including speech, handwriting and human action recognition, automatic translation, robot control etc. One of the RNN application is time series prediction used in analysis of business and financial data, anomaly detection, weather forecast. A large number of different architectures were discussed recently and the flow of new modifications of standard RNN continues to increase and all these architectures share some common features inherited from the basic systems.

Trajectory prediction based on incomplete or noisy data is one of the most amazing features of organism brains that allows living creatures to survive in complex and mostly unfriendly environment. A large number of mathematical algorithms developed for this purpose have many applications in multiple engineering field, e.g., development of guidance systems, self-driving vehicles, motor control etc. [1].

It was shown that when the input signal represents a chaotic dynamics (in discrete or discretized continuous setting) RNNs indeed predict chaotic attractor for some number of steps and then the predicted trajectories diverge from the actual ones [2–4]. This result seems natural as it reflects an important property of chaotic dynamics – extremely high sensitivity of chaotic systems to small perturbations in initial conditions.

What does happen when a trajectory is perturbed by external noise of specific statistics, e.g., white noise? How would RNN extrapolate the input of such noisy time series? Generally speaking, when a noisy signal is used as an input to a *predictive* RNN it is expected that a trained network would be able to extrapolate the *noisy* time series. It appeared that the extrapolated trajectory is not noisy – filtering of the noisy perturbation of the Lorenz attractor dynamics was reported in [5] where the authors used recurrent multi-layer perception network and noted that the reconstructed signals were ”reasonably close to the noise-free signal and the iterated predictions are smoother in comparison to the noisy signals” [5]. This observation leads to the following question - given a smooth deterministic function with added noise component as a RNN input will the trajectory anticipated by RNN be noisy or smooth? A short note [6] considered LSTM network [7] with 128 neurons trained on the Mackey-Glass time series with added noise and demonstrated that with the increase of the noise level LSTM behaviour depends more on its own dynamics than on the input data. On the contrary, the training using the noiseless input produces RNN with very high sensitivity to small perturbations.

In this manuscript we attempt to explain the fact that RNN trained on segments of *noisy* trajectory and being fed a segment of such trajectory generates a *smooth* extrapolating curve. Our analysis shows that smooth predictions are commonplace and independent of the RNN type or extrapolation quality. We establish conditions for such RNN behavior and find that when these conditions are met a new very fast predictive algorithm can be implemented. We demonstrate that this algorithm for relatively long input sequences (around 100 time points) works as good as the original one and gives the speed up to two orders of magnitude.

The manuscript is organized as follows. Section 2 describes the architecture of a very simple network made of a single recurrent network with small number of neurons followed by a linear layer. Section 3 describes RNN governing transformations and presents a standard algorithm used for time series prediction. Next Section 4 deals with the network training and discusses the dependence of the prediction quality on the number of neurons in RNN. Section 5 considers the input noise influence onto RNN state dynamics and demonstrates that it cannot be neglected. Then in Section 6 the focus shifts to the RNN dynamics during a recursive prediction procedure and conditions when this procedure results in smooth output are established. We show that satisfaction of these conditions allows to design a new much faster predictive algorithm described in details in Section 7 and demonstrate its high quality of extrapolation. The next Section 8 is devoted to possible implications of the presented results for neurosciences. Section 9 is devoted to discussion of possible applications and generalizations of our findings.

2 Network architecture and predictive algorithm

Consider a simple two layer network designed to predict multidimensional time series $\boldsymbol{\mathcal{X}} = \{\boldsymbol{x}_i\}$, $1 \le i \le N$. The first layer is a recursive network with n neurons – it takes a subsequence $\boldsymbol{X}_{k,m} = \{\boldsymbol{x}_i\} = \{\boldsymbol{x}_{k+1}, \boldsymbol{x}_{k+2}, \ldots, \boldsymbol{x}_{k+m}\}$, $0 \le k \le N - m$, of m vectors $\boldsymbol{x}_i$ having dimension d each and returns a sequence $\boldsymbol{S}$ of n-dimensional state vectors $\boldsymbol{s}_i$, $(1 \le i \le m)$. The last element $\boldsymbol{s}_m$ is transferred into the second linear layer that generates an output vector $\bar{\boldsymbol{x}}$ of dimension d by linear transformation $\bar{\boldsymbol{x}} = \boldsymbol{W} \cdot \boldsymbol{s}_m + \boldsymbol{b}$, with matrix $\boldsymbol{W}$ of dimensions $d \times n$ and d-dimensional bias vector $\boldsymbol{b}$.

A trained network is used for time series prediction recursively. Namely, one starts with a sequence $\boldsymbol{X}^1 = \boldsymbol{X}_{k,m}$ of length m supplied as input to the RNN; the resulting output is considered as a prediction of the next time point $\bar{\boldsymbol{x}}_{k+m+1}$ of the input sequence. The next input sequence $\boldsymbol{X}^2$ to RNN is produced by dropping the first point of $\boldsymbol{X}^1$ and adding the predicted point to the result: $\boldsymbol{X}^2 = \boldsymbol{X}_{k+1,m-1} \cup \bar{\boldsymbol{X}}_{k+m,1}$; here $\cup$ denotes union of two sequences with order of elements preserved. This sequence is used as input to the RNN that generates $\bar{\boldsymbol{x}}_{k+m+2}$ and a next input $\boldsymbol{X}^3 = \boldsymbol{X}_{k+2,m-2} \cup \bar{\boldsymbol{X}}_{k+m,2}$ is formed. Thus at j-th predictive step ($j \le m$) the input $\boldsymbol{X}_k^j$ to RNN is formed as $\boldsymbol{X}^j = \boldsymbol{X}_{k+j-1,m-j+1} \cup \bar{\boldsymbol{X}}_{k+m,j-1}$, while for $j > m$ the input is formed by the already predicted values only $\boldsymbol{X}^j = \bar{\boldsymbol{X}}_{k+j-m-1,m}$. The recursive procedure is repeated p times to produce p new time points $\bar{\boldsymbol{x}}_{k+m+i}, (1 \le i \le p)$ approximating the time series $\boldsymbol{\mathcal{X}}$ segment $\{\boldsymbol{x}_i\}$ for $k+m+1 \le i \le k+m+p$ (Figure 1). As the offset value k determining the initial point of the input sequence $\boldsymbol{X}^1$ is arbitrary but fixed for given predictive procedure, without loss of generality we further set it equal to zero. The described algorithm can be called a *moving window* prediction as it is characterized by recurrent usage of the input sequence $\boldsymbol{X}^j$ obtained from $\boldsymbol{X}^{j-1}$ by shifting one position to the right. It is easy to see that the procedure uses a double recursion – the inner one used m times in the recurrent layer and the outer is employed p times to generate the output points, so that the total number of recursions is mp.

3 Network state dynamics

In this manuscript we perform the analysis of all standard recurrent networks – basic, gated and LSTM RNNs. Consider an inner dynamics of a recurrent network in more details. The input sequence $\boldsymbol{X} = \{\boldsymbol{x}_i\}, 1 \le i \le m$ produces the network state sequence $\boldsymbol{S} = \{\boldsymbol{s}_i\}$ for the basic network

$$\boldsymbol{s}_i = \tanh(\boldsymbol{W}_{ix} \cdot \boldsymbol{x}_i + \boldsymbol{W}_{is} \cdot \boldsymbol{s}_{i-1} + \boldsymbol{b}_i), \tag{1}$$

where $\boldsymbol{W}_{ix},\ \boldsymbol{W}_{is}$ are matrices and $\boldsymbol{b}_i$ is a bias vector. The gated network [8] is governed by the followng relations

$$
\begin{aligned}
\boldsymbol{i}_i &= \sigma(\boldsymbol{W}_{ix}\cdot\boldsymbol{x}_i+\boldsymbol{W}_{is}\cdot\boldsymbol{s}_{i-1}+\boldsymbol{b}_i),\\
\boldsymbol{r}_i &= \sigma(\boldsymbol{W}_{rx}\cdot\boldsymbol{x}_i+\boldsymbol{W}_{rs}\cdot\boldsymbol{s}_{i-1}+\boldsymbol{b}_r),\\
\boldsymbol{m}_i &= \tanh(\boldsymbol{W}_{mx}\cdot\boldsymbol{x}_i+\boldsymbol{r}_i\otimes\boldsymbol{W}_{ms}\cdot\boldsymbol{s}_{i-1}+\boldsymbol{b}_m),\\
\boldsymbol{s}_i &= (1-\boldsymbol{i}_i)\otimes\boldsymbol{m}_i+\boldsymbol{i}_i\otimes\boldsymbol{s}_{i-1},\quad \boldsymbol{a}\otimes\boldsymbol{b}=\sum_k a_k b_k,
\end{aligned}
\tag{2}
$$

where $\sigma(x)=1/(1+\exp(-x))$ is the logistic sigmoid function, $\otimes$ denotes the elementwise multiplication of two vectors of the same length and the initial state $\boldsymbol{s}_0=\boldsymbol{0}$. The vectors $\boldsymbol{i}_i,\ \boldsymbol{r}_i,\ \boldsymbol{m}_i$ denote the input, reset and memory gate state respectively.

For LSTM network [7] the governing transformation that determines network state $\boldsymbol{S}=\{\boldsymbol{s}_i\}$ and cell state $\boldsymbol{C}=\{\boldsymbol{c}_i\}$ sequences is defined by

$$
\begin{aligned}
\boldsymbol{s}_i &= \boldsymbol{o}_i\otimes\tanh\boldsymbol{c}_i,\\
\boldsymbol{c}_i &= \boldsymbol{f}_i\otimes\boldsymbol{c}_{i-1}+\boldsymbol{i}_i\otimes\boldsymbol{m}_i,\\
\boldsymbol{o}_i &= \sigma(\boldsymbol{W}_{ox}\boldsymbol{x}_i+\boldsymbol{W}_{os}\boldsymbol{s}_{i-1}+\boldsymbol{b}_o),\\
\boldsymbol{i}_i &= \sigma(\boldsymbol{W}_{ix}\boldsymbol{x}_i+\boldsymbol{W}_{is}\boldsymbol{s}_{i-1}+\boldsymbol{b}_i),\\
\boldsymbol{f}_i &= \sigma(\boldsymbol{W}_{fx}\boldsymbol{x}_i+\boldsymbol{W}_{fs}\boldsymbol{s}_{i-1}+\boldsymbol{b}_f),\\
\boldsymbol{m}_i &= \tanh(\boldsymbol{W}_{mx}\boldsymbol{x}_i+\boldsymbol{W}_{ms}\boldsymbol{s}_{i-1}+\boldsymbol{b}_m),
\end{aligned}
\tag{3}
$$

where the initialization value of state $\boldsymbol{s}_0$ and cell state $\boldsymbol{c}_0$ vector is zero vector of length n. With $a=i,f,m,o$ we denote $\boldsymbol{W}_{ax},\ \boldsymbol{W}_{as}$ matrices and $\boldsymbol{b}_a$ bias vectors for the input, forget, memory and output gates respectively; all these structures are trainable and in the trained network their elements are real valued constants.

The shorthand form of the transformations (1-3) reads $\boldsymbol{s}_i=\boldsymbol{\mathcal{F}}(\boldsymbol{x}_i,\boldsymbol{s}_{i-1},\boldsymbol{P})$, where $\boldsymbol{P}$ denotes elements of all matrices and bias vectors in (1-3) and $\boldsymbol{s}_0$ is n-dimensional zero vector. As in trained network the set $\boldsymbol{P}$ is fixed we will drop it from the list of arguments of the vector function $\boldsymbol{\mathcal{F}}$

$$\boldsymbol{s}_i=\boldsymbol{\mathcal{F}}(\boldsymbol{x}_i,\boldsymbol{s}_{i-1}). \tag{4}$$

It is important to note that the governing transformations imply for every step i in (4) all components of $\boldsymbol{s}$ satisfy a condition $|s_k|\le 1, 1\le k\le n$. The equations (1-3) are accompanied by a linear transformation

$$\bar{\boldsymbol{x}}_{m+1}=\boldsymbol{W}\cdot\boldsymbol{s}_m+\boldsymbol{b}, \tag{5}$$

where $\bar{\boldsymbol{x}}_{m+1}$ is a value predicted by RNN based on the input $\boldsymbol{X}$.

4 RNN training and performance

The RNNs we use in the simulation have a small number n of neurons in the recurrent layer $1\le n\le 20$. The training set is constructed by merging 6000 segments of variable length ($5\le m\le 150$) of two periodic one-dimensional ($d=1$) functions – the sine wave $g_0(t)=\sin(2\pi t)$ and the shifted triangle wave $h_0(t)=1/2+1/\pi\arcsin(\sin 2\pi x)$. The white noise with the amplitude $a=0.15$ is added to both functions – $g(t)=g_0(t)+a\xi(t),\ h(t)=h_0(t)+a\xi(t)$. The time step Δt between the adjacent time points is selected equal to $\Delta t=0.01$. The RNNs are trained for 50 epochs on the complete set of 12000 segments with 20% validation set using Adam algorithm. The RNNs fail to predict the noisy dynamics of $g(t)$ or $h(t)$, instead all RNNs produce some smooth predictions $G_0(t)$ and $H_0(t)$, respectively. We define the quality function of prediction $F(t)$ vs. the actual dynamics $f(t)$ ($f=g,h$ and $F=G,H$) as

$$Q^{-1}=\frac{1}{p}\sum_{i=1}^{p}\|F(t_i)-f(t_i)\|^2,$$

where p is the length of the predicted sequence and $\|\ \|$ denotes the Euclidean norm.

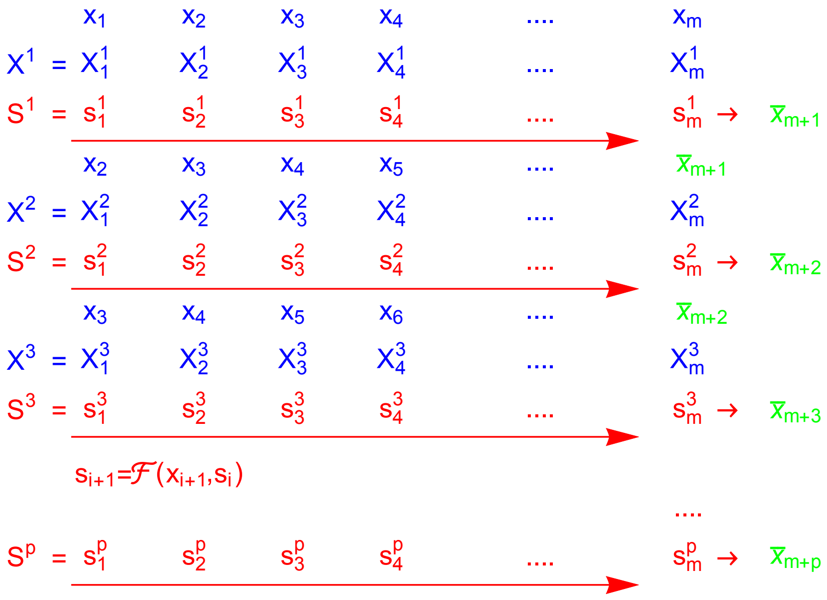

x1 x2 x3 x4 .... xm
X1 = X11 X12 X13 X14 .... X1m
S1 = s11 s12 s13 s14 .... s1m → x̄m+1
x2 x3 x4 x5 .... x̄m+1
X2 = X21 X22 X23 X24 .... X2m
S2 = s21 s22 s23 s24 .... s2m → x̄m+2
x3 x4 x5 x6 .... x̄m+2
X3 = X31 X32 X33 X34 .... X3m
S3 = s31 s32 s33 s34 .... s3m → x̄m+3
si+1=ℱ(xi+1,si)
....
Sp = sp1 sp2 sp3 sp4 .... spm → x̄m+p

Figure 1: The scheme of the prediction double recursive procedure for RNN. Three first and the last prediction steps are shown. The elements of the input sequences $\boldsymbol{X}^j$ to RNN (blue) are fed into (4) to produce recursively recurrent network states $\boldsymbol{s}_i^j$ (red). The last element $\boldsymbol{s}_m^j$ in $\boldsymbol{S}^j$ is transformed by (5) to generate the predicted point $\bar{\boldsymbol{x}}_{m+j+1}$ (shown in green). This point is used to update the input sequence $\boldsymbol{X}^{j+1}$ for the next prediction step.

As it was expected the value of Q for the LSTM network increases with n (see Figure 2). Nevertheless the predicted dynamics is always smooth which implies that the filtering property of RNN is independent of the prediction quality. We observe that for $n = 10$ the deviation of the predicted curve from the actual one is quite small for one period ($Q > 30$). Note that the prediction of the underlying smooth function was very good for $n = 20$ neurons ($Q > 100$) which is much smaller than ($n = 128$) reported in [6]. Qualitatively similar results are obtained for the basic recurrent network.

5 Noise propagation in recurrent network

Consider the process of state vector computation assuming that the input sequence $\boldsymbol{X}$ represents time point values of the function $\boldsymbol{g}(t) = \boldsymbol{g}_0(t) + a\boldsymbol{\xi}(t)$ where $\boldsymbol{g}_0(t)$ is a smooth function, $\boldsymbol{\xi}$ is a white noise random process with a small amplitude $0 \leq a \ll 1$. This implies that RNN is trained to predict the values $\boldsymbol{x}_i = \boldsymbol{g}_0(t_i) + a\boldsymbol{\xi}(t_i)$ for $i > m$ using the input $\boldsymbol{X}_m$. As the parameters of the state transformations are constants one expects that the values $\boldsymbol{s}_i$ for $i > 0$ might contain a noisy component and that eventually a sequence $\bar{\boldsymbol{X}}_{m,p}$ of the predicted values would be a representation of some noisy function. In other words, RNN is expected to produce a discrete representation of a function $\boldsymbol{G}(t)$ that mimics with some accuracy the *noisy* function $\boldsymbol{g}(t)$ using the *noisy* input $\boldsymbol{X}_m$ representing the same function $\boldsymbol{g}(t)$.

Consider step by step computation of $\boldsymbol{s}_i$. Using smallness of the noise amplitude a we find for $\boldsymbol{s}_1$ from (4) using Taylor expansion in a in linear approximation

$$\boldsymbol{s}_1 = \boldsymbol{\mathcal{F}}(\boldsymbol{g}_0(t_1) + a\boldsymbol{\xi}_1, \boldsymbol{0}) \approx \boldsymbol{\mathcal{F}}(\boldsymbol{g}_0(t_1), \boldsymbol{0}) + a\boldsymbol{\mathcal{F}}'(\boldsymbol{g}_0(t_1), \boldsymbol{0}) \otimes \boldsymbol{\eta}_1 = \hat{\boldsymbol{s}}_1 + a\tilde{\boldsymbol{s}}_1 \otimes \boldsymbol{\eta}_1, \tag{6}$$

where $\boldsymbol{\eta}$ is a n-dimensional random process obtained by a linear transformation of the d-dimensional random

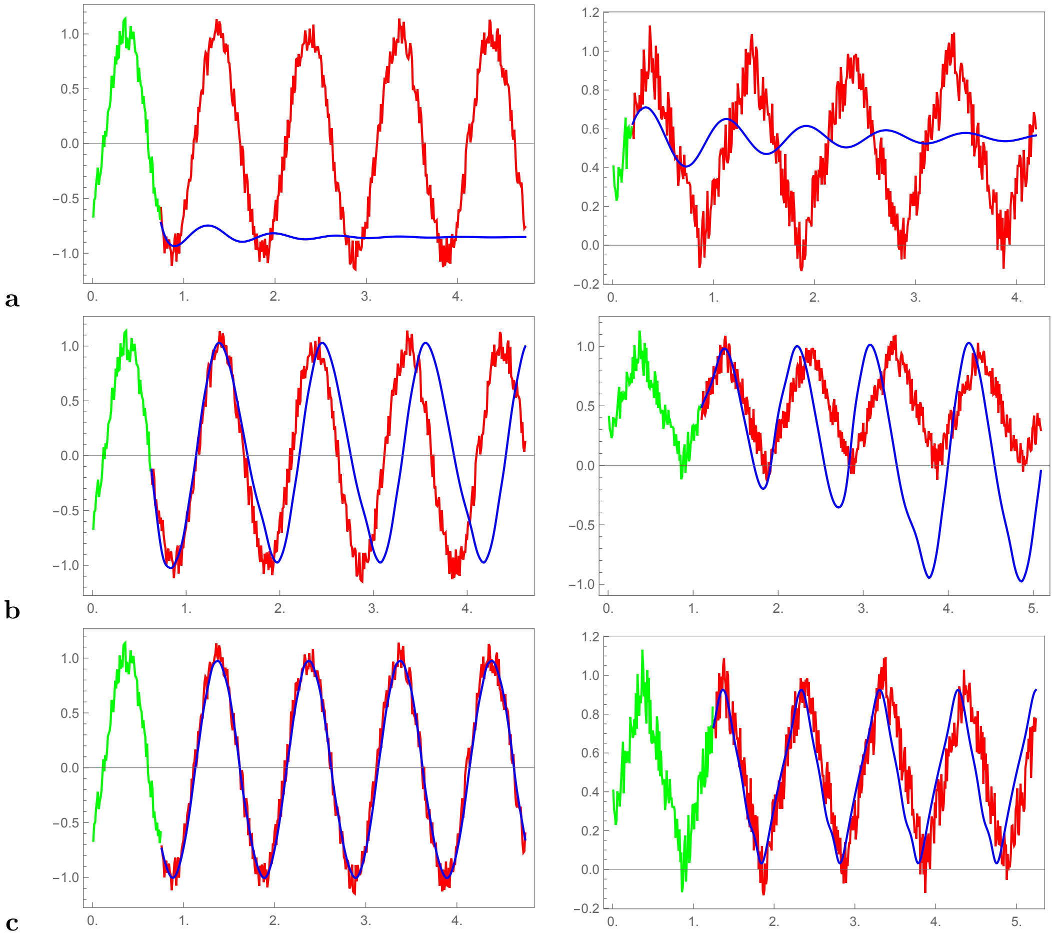


Figure 2: The input segment of the noisy ($a = 0.15$) sequence (green) of sine (left) and triangle (right) waves, the subsequent segment of $\boldsymbol{\mathcal{X}}$ (red) and predicted dynamics (blue) for (**a**) 5, (**b**) 10, (**c**) 20 neurons in LSTM network.

process $\boldsymbol{\xi}$. The computation of $\boldsymbol{s}_2$ gives

$$\begin{aligned} \boldsymbol{s}_2 &= \boldsymbol{\mathcal{F}}(\boldsymbol{g}_0(t_2) + a\boldsymbol{\xi}_2, \hat{\boldsymbol{s}}_1 + a\tilde{\boldsymbol{s}}_1 \otimes \boldsymbol{\eta}_1) \\ &\approx \boldsymbol{\mathcal{F}}(\boldsymbol{g}_0(t_2), \hat{\boldsymbol{s}}_1) + a\boldsymbol{\mathcal{F}}'(\boldsymbol{g}_0(t_2), \hat{\boldsymbol{s}}_1) \otimes (\boldsymbol{\eta}_2 + \bar{\boldsymbol{W}} \cdot \tilde{\boldsymbol{s}}_1 \otimes \boldsymbol{\eta}_1) \\ &= \boldsymbol{\mathcal{F}}(\boldsymbol{g}_0(t_2), \hat{\boldsymbol{s}}_1) + a\boldsymbol{\mathcal{F}}'(\boldsymbol{g}_0(t_2), \hat{\boldsymbol{s}}_1) \otimes \boldsymbol{\zeta}_2 = \hat{\boldsymbol{s}}_2 + a\tilde{\boldsymbol{s}}_2 \otimes \boldsymbol{\zeta}_2, \\ & \quad \boldsymbol{\zeta}_2 = \boldsymbol{\eta}_2 + \bar{\boldsymbol{W}} \cdot \tilde{\boldsymbol{s}}_1 \otimes \boldsymbol{\eta}_1, \end{aligned} \tag{7}$$

where $\bar{\boldsymbol{W}}$ denotes a matrix used in transformation of the noise component generated in the vector $\boldsymbol{s}_1$.

The subsequent steps ($1 \leq k \leq m$) produce

$$\begin{aligned} \boldsymbol{s}_k &= \boldsymbol{\mathcal{F}}(\boldsymbol{g}_0(t_k) + a\boldsymbol{\xi}_k, \hat{\boldsymbol{s}}_{k-1} + a\bar{\boldsymbol{s}}_{k-1} \otimes \boldsymbol{\zeta}_{k-1}) \\ &\approx \boldsymbol{\mathcal{F}}(\boldsymbol{g}_0(t_k), \hat{\boldsymbol{s}}_{k-1}) + a\boldsymbol{\mathcal{F}}'(\boldsymbol{g}_0(t_k), \hat{\boldsymbol{s}}_{k-1}) \otimes (\boldsymbol{\eta}_k + \bar{\boldsymbol{W}} \cdot \tilde{\boldsymbol{s}}_{k-1} \otimes \boldsymbol{\zeta}_{k-1}) \\ &= \boldsymbol{\mathcal{F}}(\boldsymbol{g}_0(t_k), \hat{\boldsymbol{s}}_{k-1}) + a\boldsymbol{\mathcal{F}}'(\boldsymbol{g}_0(t_k), \hat{\boldsymbol{s}}_{k-1}) \otimes \boldsymbol{\zeta}_k = \hat{\boldsymbol{s}}_k + a\tilde{\boldsymbol{s}}_k \otimes \boldsymbol{\zeta}_k, \\ & \quad \boldsymbol{\zeta}_k = \boldsymbol{\eta}_k + \bar{\boldsymbol{W}} \cdot \tilde{\boldsymbol{s}}_{k-1} \otimes \boldsymbol{\zeta}_{k-1}, \end{aligned} \tag{8}$$

where

$$\hat{\boldsymbol{s}}_k = \boldsymbol{\mathcal{F}}(\boldsymbol{g}_0(t_k), \hat{\boldsymbol{s}}_{k-1}), \quad \tilde{\boldsymbol{s}}_k = \boldsymbol{\mathcal{F}}'(\boldsymbol{g}_0(t_k), \hat{\boldsymbol{s}}_{k-1}),$$

and the derivative is taken w.r.t. noise amplitude a. Note that (8) is valid for $k = 1, 2$ if one defines $\boldsymbol{\zeta}_1 = \boldsymbol{\eta}_1 + \tilde{\boldsymbol{s}}_0 \otimes \boldsymbol{\zeta}_0$, and $\tilde{\boldsymbol{s}}_0$ as zero vector.

From (8) it follows that the last element $\boldsymbol{s}_m$ of the state sequence also has the noise contribution $a\tilde{\boldsymbol{s}}_m \otimes \boldsymbol{\zeta}_m$ which naturally transfers to the first predicted value

$$\bar{\boldsymbol{x}}_{m+1} = \boldsymbol{W} \cdot \hat{\boldsymbol{s}}_m + \boldsymbol{b} + a\boldsymbol{W} \cdot \tilde{\boldsymbol{s}}_m \otimes \boldsymbol{\zeta}_m = \boldsymbol{G}(t_{m+1}) = \boldsymbol{G}_0(t_{m+1}) + a\boldsymbol{W} \cdot \tilde{\boldsymbol{s}}_m \otimes \boldsymbol{\zeta}_m,$$

where $\boldsymbol{G}$ and $\boldsymbol{G}_0$ are approximations to the functions $\boldsymbol{g}$ and $\boldsymbol{g}_0$ generated by RNN. This means that the RNN itself only transforms the input noise but cannot filter it out.

The predicted element $\bar{\boldsymbol{x}}_{m+1}$ is used as the last element of the input sequence in the next prediction step and therefore one expects that the predicted sequence $\bar{\boldsymbol{X}}_{m,p}$ should reflect the noise components contained both in the input and predicted sequences. Unexpectedly, the numerical experiments (see below) show that in fact the predicted sequence $\bar{\boldsymbol{X}}_{m,p}$ is not noisy but represents the approximation $\boldsymbol{G}_0(t)$ of the smooth function $\boldsymbol{g}_0(t)$. The goal of this manuscript is to explain this unexpected behavior and to determine conditions required for generation of a smooth prediction.

6 RNN state dynamics

In the previous Section we observe that the noise component of the input signal is preserved in the RNN states, and we have to look at state dynamics in more details to understand noise filtering in the trajectory prediction process.

6.1 Numerical experiments

Consider in details the sequence of the RNN states $\boldsymbol{S}^1$ and $\boldsymbol{S}^2$ for first and second prediction steps for three values of the noise amplitude $a = 0, 0.15, 0.9$ of the input sequence. Figure 3a demonstrates that indeed the dynamics of LSTM state is affected by noise as predicted by (8). We also note that both sequences $\boldsymbol{S}^1$ and $\boldsymbol{S}^2$ look very similar. To test this similarity we overlay the corresponding sequences for given noise amplitude (Figure 3b-d) and find that even in case of large noise $a = 0.9$ the sequence $\boldsymbol{S}^2$ is very close to the sequence $\boldsymbol{S}^1$ shifted by one step to the left, in other words $\boldsymbol{s}_i^2 \approx \boldsymbol{s}_{i+1}^1$.

6.2 Dynamics of state vector shifted difference

To understand this behavior recall a relation between the input sequences $\boldsymbol{X}^j$ and $\boldsymbol{X}^{j+1}$ (see Figure 1). The input sequence $\boldsymbol{X}^j$ construction algorithm described in Section 3 implies that $\boldsymbol{X}_i^{j+1} = \boldsymbol{X}_{i+1}^j$ for all $2 \le i \le m-1$. Using (4) we find

$$\begin{aligned} \boldsymbol{s}_{i+1}^1 &= \boldsymbol{\mathcal{F}}(\boldsymbol{X}_{i+1}^1, \boldsymbol{s}_i^1), & 0 \le i \le m-1, \quad (9)\\ \boldsymbol{s}_i^2 &= \boldsymbol{\mathcal{F}}(\boldsymbol{X}_i^2, \boldsymbol{s}_{i-1}^2) = \boldsymbol{\mathcal{F}}(\boldsymbol{X}_{i+1}^1, \boldsymbol{s}_{i-1}^2), & 1 \le i \le m-1. \quad (10)\end{aligned}$$

We observe that in computation of $\boldsymbol{s}_{i+1}^1$ and $\boldsymbol{s}_i^2$ the first argument of the map $\boldsymbol{\mathcal{F}}$ in (9,10) is the same. Consider the difference $\boldsymbol{\delta}_i^1 = \boldsymbol{s}_{i+1}^1 - \boldsymbol{s}_i^2$. For $i = 0$ we have $\boldsymbol{\delta}_1^1 = \boldsymbol{s}_1^1 = \boldsymbol{\mathcal{F}}(\boldsymbol{X}_1^1, \boldsymbol{0})$. For $i = 1$ find

$$\boldsymbol{\delta}_1^1 = \boldsymbol{s}_2^1 - \boldsymbol{s}_1^2 = \boldsymbol{\mathcal{F}}(\boldsymbol{X}_2^1, \boldsymbol{s}_1^1) - \boldsymbol{\mathcal{F}}(\boldsymbol{X}_2^1, \boldsymbol{0}) = \boldsymbol{\mathcal{F}}(\boldsymbol{X}_2^1, \boldsymbol{\delta}_0^1) - \boldsymbol{\mathcal{F}}(\boldsymbol{X}_2^1, \boldsymbol{0}).$$

Assuming $\|\boldsymbol{\delta}_0^1\| \ll 1$ expand the first term above and retain the leading order to obtain

$$\boldsymbol{\delta}_1^1 = \frac{\partial \boldsymbol{\mathcal{F}}(\boldsymbol{X}_2^1, \boldsymbol{s} = \boldsymbol{0})}{\partial \boldsymbol{s}} \cdot \boldsymbol{\delta}_0^1 = \boldsymbol{A}_1^1 \cdot \boldsymbol{\delta}_0^1. \tag{11}$$

With $i = 2$ find

$$\boldsymbol{\delta}_2^1 = \boldsymbol{s}_3^1 - \boldsymbol{s}_2^2 = \boldsymbol{\mathcal{F}}(\boldsymbol{X}_3^1, \boldsymbol{s}_2^1) - \boldsymbol{\mathcal{F}}(\boldsymbol{X}_3^1, \boldsymbol{s}_1^2) = \boldsymbol{\mathcal{F}}(\boldsymbol{X}_3^1, \boldsymbol{s}_1^2 + \boldsymbol{\delta}_1^1) - \boldsymbol{\mathcal{F}}(\boldsymbol{X}_3^1, \boldsymbol{s}_1^2),$$

and the expansion leads to

$$\boldsymbol{\delta}_2^1 = \frac{\partial \boldsymbol{\mathcal{F}}(\boldsymbol{X}_3^1, \boldsymbol{s} = \boldsymbol{s}_1^2)}{\partial \boldsymbol{s}} \cdot \boldsymbol{\delta}_1^1 = \boldsymbol{A}_2^1 \cdot \boldsymbol{\delta}_1^1 = \boldsymbol{A}_1^1 \cdot \boldsymbol{A}_2^1 \cdot \boldsymbol{\delta}_0^1. \tag{12}$$

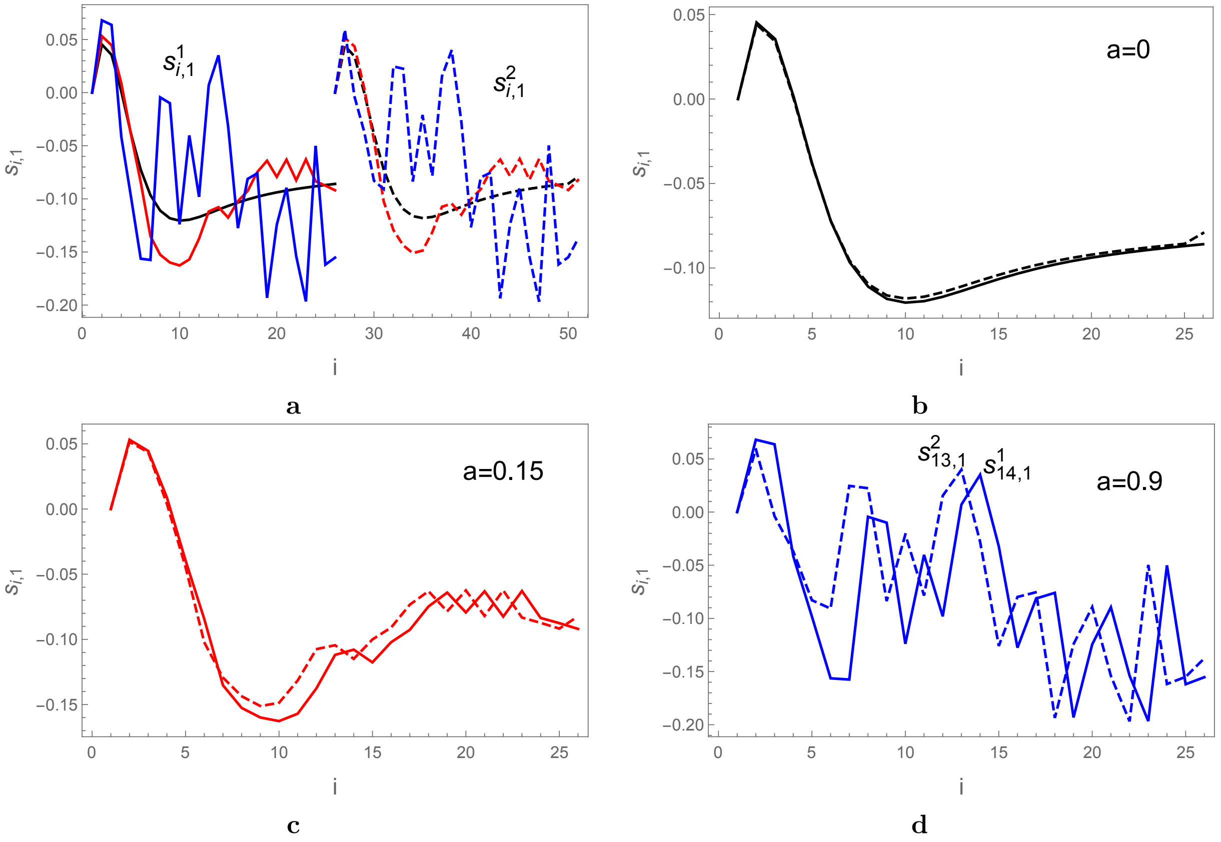


Figure 3: The dynamics of the first element $s^j_{i,1}$ of the state vector $\boldsymbol{s}^j_i$ in the j-th round of prediction for $j=1$ (solid) and $j=2$ (dashed) for three noise amplitudes – $a=0$ (black), $a=0.15$ (red) and $a=0.9$ (blue). (**a**) The sequence $s^2_{i,1}$ is shifted w.r.t. of $s^1_{i,1}$. (**b - d**) The sequences are overlapped for different noise amplitudes: (**b**) $a=0$ (no noise), (**c**) original amplitude $a=0.15$, (**d**) increased amplitude $a=0.9$. The values of $s^1_{i+1,1}$ and $s^2_{i,1}$ tend to each other with increasing i.

It is easy to deduce that for $i=m-1$

$$\boldsymbol{\delta}^1_{m-1}=\boldsymbol{A}^1\cdot\boldsymbol{\delta}^1_0,\quad \boldsymbol{A}^1=\prod_{k=0}^{m-1}\boldsymbol{A}^1_k,\quad \boldsymbol{A}^1_k=\frac{\partial\boldsymbol{\mathcal{F}}(\boldsymbol{X}^1_{k+1},\boldsymbol{s}=\boldsymbol{s}^2_{k-1})}{\partial\boldsymbol{s}}. \tag{13}$$

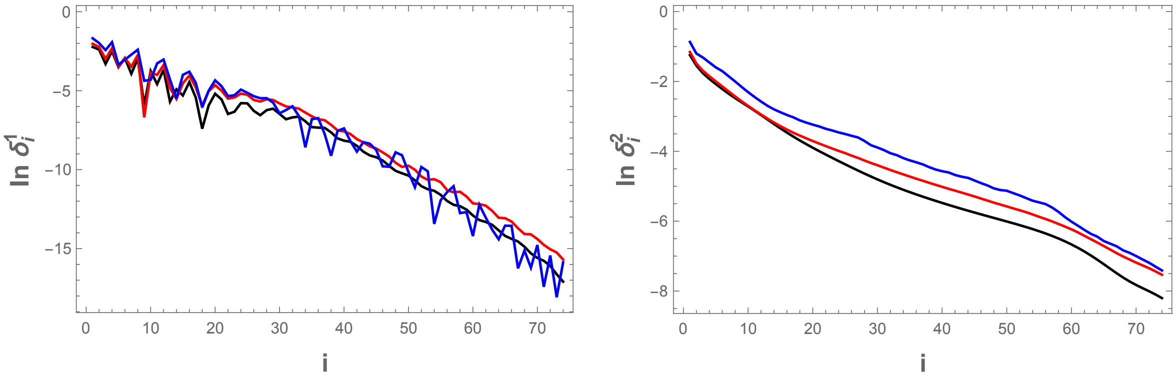
Generalizing the above relations to the other rounds of the predictive cycle we obtain for $\boldsymbol{\delta}^j_{m-1}=\boldsymbol{s}^j_m-\boldsymbol{s}^{j+1}_{m-1}$:

$$\boldsymbol{\delta}^j_{m-1}=\boldsymbol{A}^j\cdot\boldsymbol{\delta}^j_0,\quad \boldsymbol{A}^j=\prod_{k=0}^{m-1}\boldsymbol{A}^j_k,\quad 1\le j\le p. \tag{14}$$

The numerical simulations of the state dynamics in the basic and gated RNNs demonstrate the exponential decay of shifted difference norm (Figure 4a,b). In Appendix we show that for the basic RNN the exponential decay of δ^j_i takes place due to a specific spectral property of the matrix $\boldsymbol{W}_{is}$, namely, the absolute value of all eigenvalues of this matrix should be less than unit.

In the LSTM network the relations similar to (9-14) are valid with respect to the cell state vectors $\boldsymbol{c}^j_i$ and one can write for $\boldsymbol{d}^j_i=\boldsymbol{c}^j_{i+1}-\boldsymbol{c}^{j+1}_i$:

$$\boldsymbol{d}^j_{m-1}=\boldsymbol{B}^j\cdot\boldsymbol{d}^j_0,\quad \boldsymbol{B}^j=\prod_{k=0}^{m-1}\boldsymbol{B}^j_k,\quad \boldsymbol{B}^j_k=\frac{\partial\boldsymbol{\mathcal{F}}(\boldsymbol{X}^j_{k+1},\boldsymbol{c}=\boldsymbol{c}^{j+1}_{k-1})}{\partial\boldsymbol{c}},\quad 1\le j\le p. \tag{15}$$



a

b

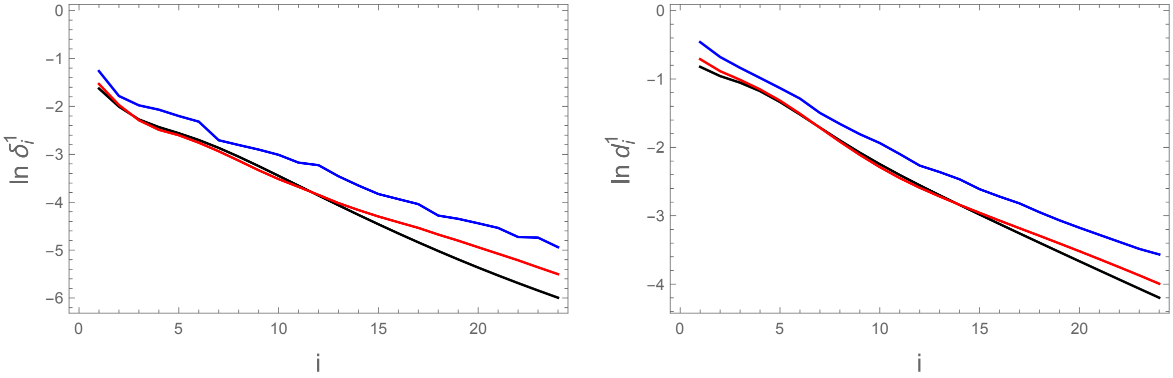
Figure 4: The dynamics of the shifted difference norm (**a**) δ_i^1 in the basic RNN with $n = 10$, (**b**) δ_i^2 in the gated RNN with $n = 20$, for the noise amplitude $a = 0$ (black), 0.15 (red) and 0.9 (blue).

Similarly, with $\det \boldsymbol{B}_i^j < 0$, a deviation norm $d_i^j = \|\boldsymbol{d}_i^j\|$ satisfies $d_i^j < d_{i-1}^j$ and would decrease exponentially. The computations for $j = 1$ show (see Figure 5) that indeed both δ_i^1 and d_i^1 decrease exponentially with i

$$\delta_i^1 = \delta_1^1 e^{-\alpha i}, \qquad d_i^1 = d_1^1 e^{-\beta i}, \tag{16}$$

and both decay rates α and β are not affected by the noise strength but depend on i, i.e., for large i they might tend to zero. It is possible that decay rates behavior also depends on the number of neurons n. The simulations show that similar behavior remains valid for all steps of the prediction procedure

$$\delta_i^j \sim e^{-\alpha i}, \qquad d_i^j \sim e^{-\beta i}, \quad 1 \le j \le p. \tag{17}$$



a

b

Figure 5: The shifted difference norms (**a**) δ_i^1 of state vectors and (**b**) d_i^1 of cell state vectors of LSTM network with $n = 10$ decay exponentially with i for the noise amplitude $a = 0$ (black), 0.15 (red) and 0.9 (blue).

This means also that the state vector $\boldsymbol{s}_{m-1}^{j+1}$ (next to last in the sequence $\boldsymbol{S}^{j+1}$) is very close to the last vector $\boldsymbol{s}_m^j$ of the preceding sequence $\boldsymbol{S}^j$, i.e.,

$$\boldsymbol{s}_{m-1}^{j+1} = \boldsymbol{s}_m^j + \boldsymbol{\epsilon}^j, \qquad \epsilon^j \ll 1. \tag{18}$$

6.3 Approximate governing transformation

Now it is time to recall that the state vector $\boldsymbol{s}_m^j$ gives rise to the prediction $\bar{\boldsymbol{x}}_{m+j} = \boldsymbol{W} \cdot \boldsymbol{s}_m^j + \boldsymbol{b}$, and this value is used as the last element of the input sequence for the next prediction step: $\boldsymbol{X}_m^{j+1} = \boldsymbol{W} \cdot \boldsymbol{s}_m^j + \boldsymbol{b}$.

Employ the relation (4) for $i = m$ to find

$$\boldsymbol{s}_m^{j+1} = \boldsymbol{\mathcal{F}}(\boldsymbol{X}_m^{j+1}, \boldsymbol{s}_{m-1}^{j+1}) = \boldsymbol{\mathcal{F}}(\boldsymbol{W} \cdot \boldsymbol{s}_m^j + \boldsymbol{b}, \boldsymbol{s}_m^j + \boldsymbol{\epsilon}^j) \approx \boldsymbol{\mathcal{F}}(\boldsymbol{W} \cdot \boldsymbol{s}_m^j + \boldsymbol{b}, \boldsymbol{s}_m^j) = \boldsymbol{\mathcal{G}}(\boldsymbol{s}_m^j). \tag{19}$$

The map $\boldsymbol{\mathcal{G}}$ for LSTM is defined by the transformations (for $j > 1$)

$$\begin{aligned} \boldsymbol{s}_m^j &= \boldsymbol{o}_m^j \otimes \tanh \boldsymbol{c}_m^j, & \boldsymbol{c}_m^j &= \boldsymbol{f}_m^j \otimes \boldsymbol{c}_m^{j-1} + \boldsymbol{i}_m^j \otimes \boldsymbol{m}_m^j, \\ \boldsymbol{o}_m^j &= \sigma(\tilde{\boldsymbol{W}}_{os} \boldsymbol{s}_m^{j-1} + \tilde{\boldsymbol{b}}_o), & \boldsymbol{i}_m^j &= \sigma(\tilde{\boldsymbol{W}}_i \boldsymbol{s}_m^{j-1} + \tilde{\boldsymbol{b}}_i), \\ \boldsymbol{f}_m^j &= \sigma(\tilde{\boldsymbol{W}}_{fs} \boldsymbol{s}_m^{j-1} + \tilde{\boldsymbol{b}}_f), & \boldsymbol{m}_m^j &= \tanh(\tilde{\boldsymbol{W}}_{ms} \boldsymbol{s}_m^{j-1} + \tilde{\boldsymbol{b}}_m), \end{aligned} \tag{20}$$

where

$$\tilde{\boldsymbol{W}}_{as} = \boldsymbol{W}_{ax} \cdot \boldsymbol{W} + \boldsymbol{W}_{as}, \quad \tilde{\boldsymbol{b}}_a = \boldsymbol{W}_{ax} \cdot \boldsymbol{b} + \boldsymbol{b}_a, \quad a = i, f, m, o, \tag{21}$$

and $\boldsymbol{s}_m^1$ and $\boldsymbol{c}_m^1$ are obtained by application of (3) to the original input sequence. It is easy to see that (20) can be obtained from (3) by setting all $\boldsymbol{W}_{ax} = \boldsymbol{0}$ and using the replacements $\boldsymbol{W}_{as} \to \tilde{\boldsymbol{W}}_{as}$ and $\boldsymbol{b}_a \to \tilde{\boldsymbol{b}}_a$ defined in (21). Similar procedure can be applied to (1) and (2) for basic and gated RNN respectively and it gives for the basic network a simple transformation

$$\boldsymbol{s}_m^j = \tanh(\tilde{\boldsymbol{W}}_{is} \cdot \boldsymbol{s}_m^{j-1} + \tilde{\boldsymbol{b}}_i). \tag{22}$$

We observe that the influence of the input sequence $\boldsymbol{X}^j$ (and the noise contained in it) on the dynamics of the RNN last state vector $\boldsymbol{s}_m^j$ is negligible and the latter is almost completely determined by the same vector $\boldsymbol{s}_m^{j-1}$ at the preceding prediction step.

7 A new fast algorithm for trajectory prediction

The main result in previous Section implies that after computation of $\boldsymbol{s}_m^1$ using m times the recursion (4) the original input sequence can be dropped and the transformation (19) is applied recursively $p-1$ times to generate $\boldsymbol{s}_m^j$ for $2 \le j \le p$. Then the linear transformation (5) produces the desired sequence $\bar{\boldsymbol{x}}_{m+j}$ for $1 \le j \le p$.

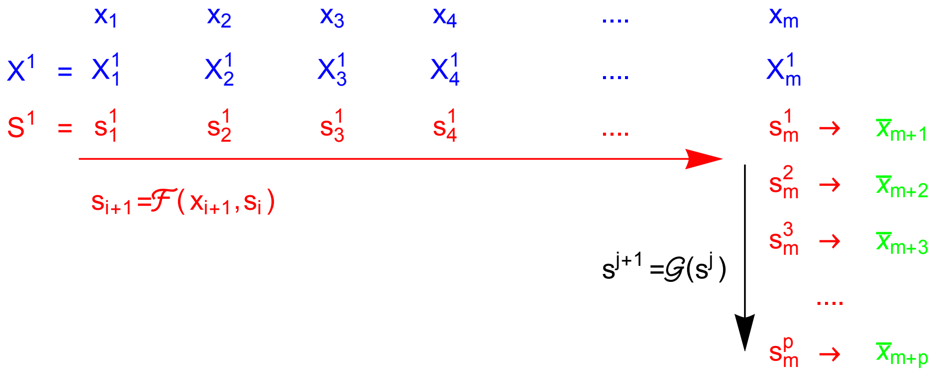
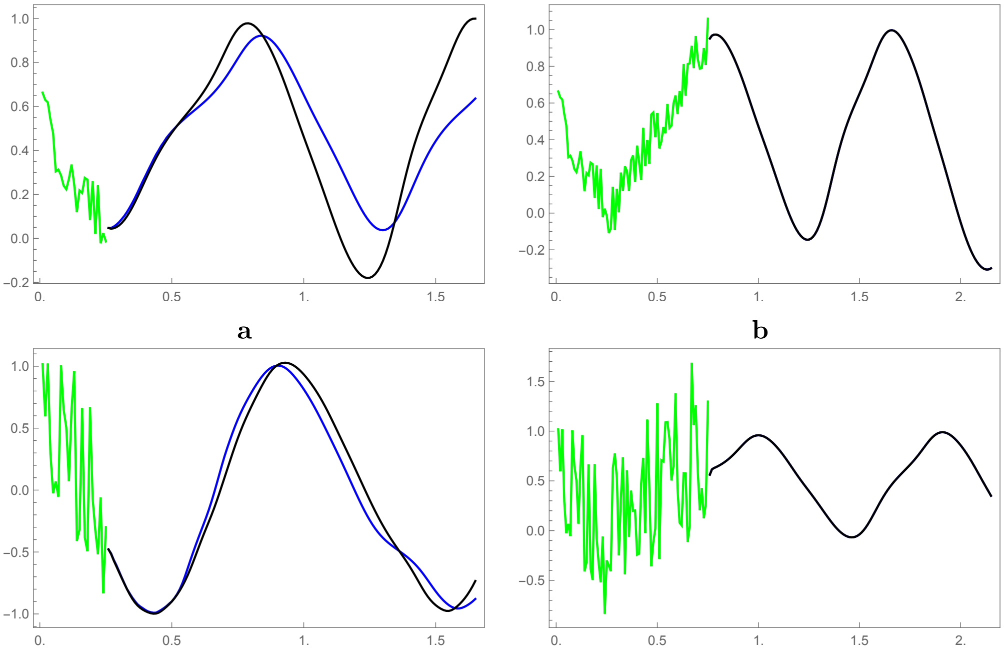


Figure 6: The approximate scheme of the recursive prediction based on (19). The standard prediction sequence (4) is evoked only once to produce $\boldsymbol{s}_m^1$ and then the approximate algorithm (19) is applied recursively to produce $\boldsymbol{s}_m^j$ (red). The predicted points $\bar{\boldsymbol{x}}_{m+j}$ (green) are computed using (5).

These steps represent a new very fast prediction algorithm (Figure 6). The transformation (19) might produce in principle non-smooth and even chaotic dynamics but nevertheless it is important that the noise component in the input sequence plays no role in the generation of the anticipated points. On the other hand this noise component can strongly affect the result of RNN training influencing the weights and biases of the trained network.

We use the approximate map (19) to compute the predicted sequence for the input of different length m and compare the results to the prediction made by iterative application of RNN. We find that increase in input sequence length m improves the approximate prediction (Figure 7) up to a perfect coincidence with the traditional approach prediction. It is explained by the fact that for large m the difference $\boldsymbol{\epsilon}^1$ becomes extremely small that increases the accuracy of the map (19). Moreover, when we increase the input sequence noise amplitude a six times compared to the value at which LSTM network was trained, the approximate procedure still generates prediction coinciding with the one produced by LSTM itself (Figure 7d).

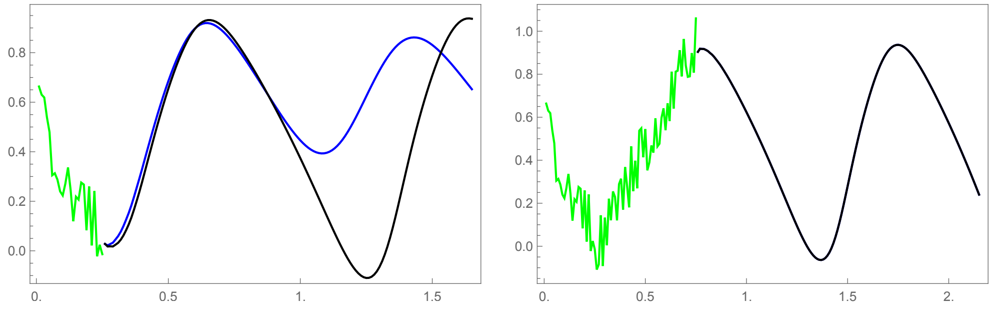


a **b**

c **d**

Figure 7: Comparison of the predictions for the trained LSTM network by the moving window procedure (blue) and by using the map (20) (black) for the triangle wave input sequence (green) with variable noise amplitude a and length m: (**a**) $a = 0.15,\ m = 25$, (**b**) $a = 0.15,\ m = 75$, (**c**) $a = 0.9,\ m = 25$, (**d**) $a = 0.9,\ m = 75$; in (**b**) and (**d**) both predictions coincide.

We also compare the predictions made by the RNN governed by (1) and (22) and find that these predictions coincide for large m (Figure 8b).



a **b**

Figure 8: Comparison of the predictions for the basic RNN with $n = 10$ by the moving window algorithm (blue) and by the map (22) (black) for the triangle wave input sequence (green) of (**a**) $m = 25$ and (**b**) $m = 75$ points with noise amplitude $a = 0.15$; in (**b**) both predictions coincide.

We observe that the moving window prediction generating p time series points using the trained RNN is a recursion (p times) each consisting of m inner recursions, i.e., total $R_o = mp$ recursion steps while the approximate procedure (19) replaces it by $R_a = m + p - 1$ recursions (Figure 6). Assuming that the computation time $\mathcal{T}$ is linearly proportional to the total recursion number $\mathcal{T} = \mu R$ estimate a speed up $\kappa = \mathcal{T}_o/\mathcal{T}_a$. The length m of the input sequence $\boldsymbol{X}$ should be quite large ($m \gg 1$) in order to generate a high quality prediction. The length $p = \gamma m$ of the predicted sequence $\bar{\boldsymbol{X}}$ is comparable to m, i.e., $\gamma \gtrapprox 1$ and we find the estimate of the prediction times ratio $\kappa = mp/(m+p) = \gamma m/(1+\gamma) > m/2$. Thus the approximate prediction algorithm gain is proportional to the length of the input sequence. We observed that $m \approx 50$ leads to high quality of the approximate scheme (Figures 7, 8) and thus one can have speed up of an order of magnitude without loss of prediction quality.

8 Algorithms robustness analysis

The results presented above can have important implications in neuroscience. If one assumes that brain uses recurrent networks for trajectory prediction and it employs the moving window procedure described in Section 2 (see Fig. 1) then the implementation of this algorithm requires satisfaction of several conditions. These include – the value (amplitude) of the input element should not change significantly during time interval when this element is used for prediction; the order of the elements of the input sequence at the second and subsequence steps of prediction should not be changed. The first condition can be broken if the signal value is perturbed by inner noise or it decays with some rate. As the influence of noise on the input sequence is shown not to be critical for the prediction we will focus on the signal decay influence of the prediction quality. The second condition is probably more difficult to meet and we have to consider a case when on each step of prediction some elements of the updated input sequence are partially reshuffled.

Consider first how the input element decay rate affects the quality of prediction. For the LSTM network

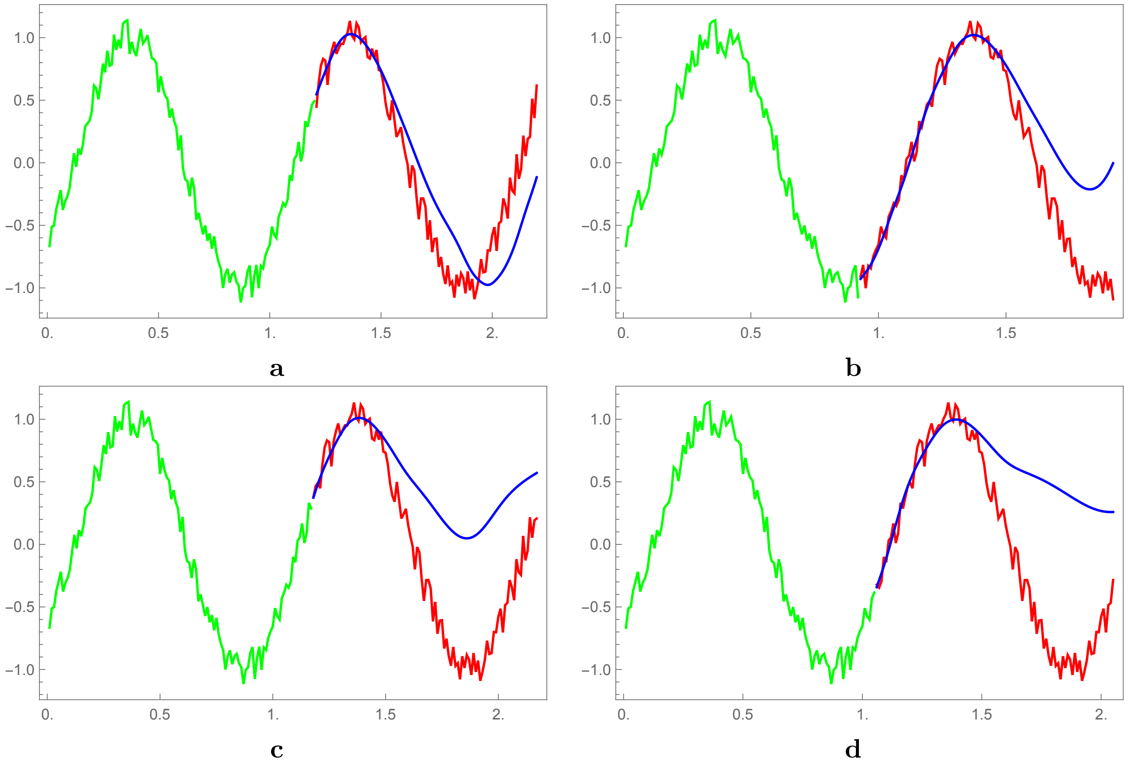


Figure 9: Comparison of the predictions by the LSTM network (blue) with $n = 10$ to the continuation (red) of the sine wave input (green) for different rate α of input values exponential decay: (**a**) 0, (**b**) 0.002, (**c**) 0.005, (**d**) 0.008.

we observe that the increase of the decay rate leads to faster deviation of the predicted trajectory from the expected one (Figure 9 a-d), nevertheless the predicted trajctory remains quite smooth.

It appears that the partial reshuffling of the input sequence at each prediction step affects not only the prediction quality but also generates nonsmooth extrapolated curves (Figure 10). We observe that satisfaction of both conditions mentioned above is critical for a successful prediction using the moving window algorithm and if any of them is not satisfied the increase of the length m of input sequence makes an accurate prediction nearly impossible. The reduced algorithm (see Figure 6) is free of these limitations as it does not require any knowledge of the input sequence $\boldsymbol{X}^j$ for $j > 1$ but instead employs the internal network dynamics, and the quality of prediction grows with the length m of the initial input sequence. Thus we conclude that the new fast algorithm appears to be much more robust compared to the traditional moving window approach.

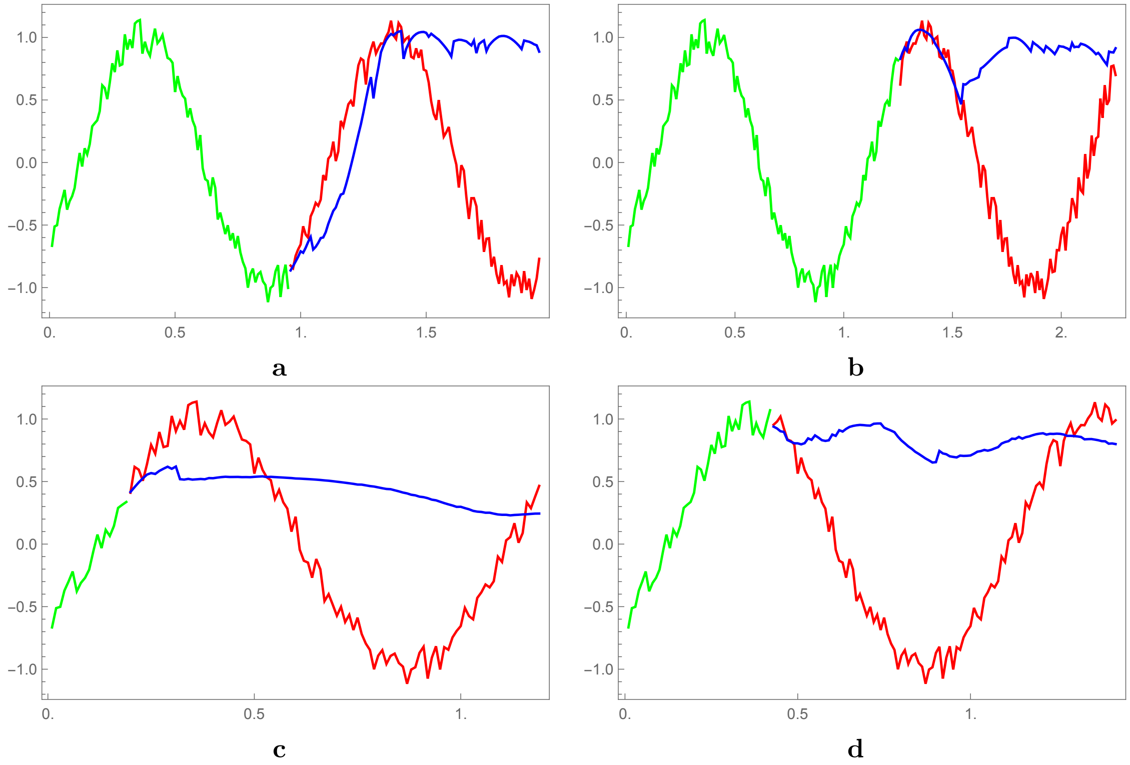


Figure 10: Comparison of the predictions by the trained LSTM network (blue) with $n = 10$ to the continuation (red) of the sine wave input (green) when the input sequence is partially reshuffled.

9 Discussion

In this manuscript we show that the predictive RNN based on a single recurrent layer with a small number of neurons works as an effective noise filter. Namely, when the RNN is supplied by the noisy input sequence of (multidimensional) time series points and used recursively for series extrapolation it generates points that belong to some smooth curve that mimics the smoothed original time series. Using the analysis of the recursive prediction procedure we established a set of conditions required to observe such behavior. These conditions imply that the governing transformation of the predictive algorithm reduces to one that requires the input sequence only once and later does not depend on it. As the result the predictive algorithm can be drastically simplified and accelerated without loss of accuracy. The overall quality of prediction strongly depends on the length of the input sequence while the acceleration is proportional to it. Thus using the approximate predictive algorithm one can both increase the quality and save time and computational resources.

These results allow to conclude that RNNs with several recurrent layers of a single or multiple types would have the same property of noise filtration off an input sequence. Moreover it is possible to suggests that any neural network of several layers would share this behavior if it has a recurrent network preceding a last layer that generates the network prediction.

The approximate predictive algorithm is governed by a multidimensional discrete map with the parameters determined by the weights and biases of the trained RNN only and does not require the input sequence. In all our numerical experiments we observe that the parameters of the trained network always lead to smooth dynamics generated by this reduced map. The same time setting these parameters to random real values sometimes produces nonsmooth and quite nontrivial dynamics including complex periodic and even chaotic trajectories. It is very important to understand what is special about the parameters of the trained network that they *always* produce smooth trajectory generated by both the original and approximate predictive schemes.

Another important aspect of RNN noise filtering is related to neuroscience. Brain ability to predict a trajectory is one of the most important requirements for survival and this natural ability is highly developed. By default the brain should be able to predict trajectories based on incomplete or noisy data, and it has to do this with high reliability. Moreover, the predictions should be made for several objects simultaneously and it requires large resources. Even if an object actual trajectory in space is smooth it is transferred into brain by the receptors as a *noisy* time series. The trajectory prediction is usually considered as a two-stage process – first, the brain performs initial classification of the trajectory and then, in case when the organism should somehow react to this specific motion, a precise predictive mechanism is activated. If the available data is noisy both these stages would require more resources compared to processing of smooth trajectories. We hypothesize that first of all activates an additional inexpensive (with small number of neurons) recurrent network. It would effectively filter noise out and transfer a cleaned smooth trajectory segment to the classification and then to precise predictive networks. Note that in this case the latter networks resources can be greatly reduced.

We also learned that the prediction process itself can be significantly accelerated by using the approximate algorithm described in the manuscript. It would be interesting to address a possibility of a physiological implementation of this scheme. If this algorithm does work in the brain the trajectory prediction is done in two stages – first the existing trajectory segment is fed into the network and the first point is predicted. Then the input information is forgotten and the brain predicts subsequent points based on the approximate scheme. We showed that the moving window prediction procedure is very sensitive to various perturbations of the input sequence during its update that might strongly reduce the prediction quality. Moreover, the longer is the input sequence the higher chance is for these perturbations to influence the result. On the other hand, the reduced fast algorithm is much more robust with respect to those perturbations and allows to reach high predictabilty which is proportional to the length of the input sequence.

One has to take into account that the number of predicted elements is usually smaller or approximately equal to the length of the input sequence as the prediction accuracy is inversly proportional to the length of predicted sequence. Thus the receptors provide a new input sequence is and a correction of predicted trajectory is performed. It saves resources and helps to resolve the problem of prediction time minimization – there exists a range of lengths m of the input sequence for which the prediction quality is proportional to m thus brain tends to increase the value of m. This increase requires a linearly proportional increase in prediction time when the moving window algorithm is employed. A switch to the approximate algorithm allows significant reduction in the processing time without loss in the prediction quality.

The existence of fast and robust predictive algorithm in recurrent networks has important implications for both brain research and artificial neural network field. The trajectory prediction is an important but is not the only brain predictive task. For example, a trained human brain can perform symbol-based predictions as finishing a word or sentence, solve simple arithmetic problems like addition and subtraction, recognize and continue a sequence of musical notes etc. In recent years a novel Transformer architecture was proposed to solve this kind of problems and these networks do not use a recurrent paradigm or convolution; instead a notion of attention is introduced to be a main element [9]. Today these networks are in the focus of research and a significant progress in this field is observed (see, for example, a description of GPT-3 Transformer in [10]). The text generation algorithm is the moving (or expanding) window recurrent procedure while the inner transformation differs from the one employed in the recurrent networks. It is possible that for the Transformer networks also exists a reduced algorithm that does not require repeated input of the updated symbol sequence at each step of prediction and instead employs an internal dynamics of the trained network. A search for such algorithms might help to explain how human brain performs language and other symbol based tasks.

10 Notation

Symbol and definition	Conditions	Meaning
$\boldsymbol{\mathcal{X}} = \{\boldsymbol{x}_i\}$	$1 \leq i \leq N$	original time series with elements $\boldsymbol{x}_i$
$\boldsymbol{x}_i = \boldsymbol{g}(t_i)$		element $\boldsymbol{x}_i$ is a value of a function $\boldsymbol{g}$ at $t = t_i$
d	$d \geq 1$	dimension of $\boldsymbol{x}_i$
$\boldsymbol{g}(t) = \boldsymbol{g}_0(t) + a\boldsymbol{\xi}(t)$		$\boldsymbol{g}$ is a sum of a smooth function $\boldsymbol{g}_0$ and noise $\boldsymbol{\xi}$
a	$a \geq 0$	noise amplitude
$\boldsymbol{X}_{k,m} = \{\boldsymbol{x}_i\}$	$k+1 \leq i \leq k+m$	segment of $\boldsymbol{\mathcal{X}}$ of the length m starting with $\boldsymbol{x}_{k+1}$
$\bar{\boldsymbol{x}}_i$		i-th element of $\boldsymbol{\mathcal{X}}$ predicted by RNN
$\bar{\boldsymbol{X}}_{k+m,p} = \{\bar{\boldsymbol{x}}_i\}$	$k+m+1 \leq i \leq k+m+p$	sequence of p predicted elements based on input $\boldsymbol{X}_k$
$\boldsymbol{X}^j$	$j > 0$	input to RNN at j-th step of recursive prediction
$\boldsymbol{S}^j = \{\boldsymbol{s}_i^j\}$	$1 \leq i \leq m$	sequence of RNN states for input $\boldsymbol{X}^j$
$\boldsymbol{s}_i^j$		state vector at j-th step of recursive prediction
n	$n \geq 1$	dimension of state vector $\boldsymbol{s}_i^r$
$\boldsymbol{W}_{ax}$	$a = i, m, f, o$	$n \times d$ matrix
$\boldsymbol{W}_{as}$	$a = i, m, f, o$	$n \times n$ matrix
$\boldsymbol{b}_a$	$a = i, m, f, o$	n-dimensional vector
$\boldsymbol{W}$		$d \times n$ matrix
$\boldsymbol{b}$		d-dimensional vector

Table 1: Symbols and corresponding definitions used in the manuscript.

Acknowledgements

The author wishes to thank Vladimir Zverev, Yuri Shvachko, Mitya Chkolvsky and Kausik Si for fruitful discussions.

Appendix

Shifted difference dynamics for basic recurrent network

The simplest RNN transformation reads

$$\boldsymbol{s}_i = f(\boldsymbol{W}_{ix} \cdot \boldsymbol{x}_i + \boldsymbol{W}_{is} \cdot \boldsymbol{s}_{i-1} + \boldsymbol{b}_i), \tag{A1}$$

where the nonlinear scalar function $f(x) = \tanh x$ is applied to all components of its vectorial argument. The shifted difference $\boldsymbol{\delta}_i^j = \boldsymbol{s}_{i+1}^j - \boldsymbol{s}_i^{j+1}$ reads

$$\begin{aligned}
\boldsymbol{\delta}_i^j &= f(\boldsymbol{W}_{is} \cdot \boldsymbol{s}_i^j + \boldsymbol{W}_{ix} \cdot \boldsymbol{x}_{i+1}^j + \boldsymbol{b}) - f(\boldsymbol{W}_{is} \cdot \boldsymbol{s}_{i-1}^{j+1} + \boldsymbol{W}_{ix} \cdot \boldsymbol{x}_i^{j+1} + \boldsymbol{b}) \\
&= f(\boldsymbol{W}_{is} \cdot \boldsymbol{s}_i^j + \boldsymbol{W}_{ix} \cdot \boldsymbol{x}_{i+1}^j + \boldsymbol{b}) - f(\boldsymbol{W}_{is} \cdot (\boldsymbol{s}_i^j - \boldsymbol{\delta}_{i-1}^j) + \boldsymbol{W}_{ix} \cdot \boldsymbol{x}_{i+1}^j + \boldsymbol{b}) \\
&= f(\boldsymbol{y}_i^j) - f(\boldsymbol{y}_i^j - \boldsymbol{W}_{is} \cdot \boldsymbol{\delta}_{i-1}^j) \qquad \boldsymbol{y}_i^j = \boldsymbol{W}_{is} \cdot \boldsymbol{s}_i^j + \boldsymbol{W}_{ix} \cdot \boldsymbol{x}_{i+1}^j + \boldsymbol{b} = f^{-1}(\boldsymbol{s}_{i+1}^j),
\end{aligned} \tag{A2}$$

where f^{-1} denotes an inverse function to f and we use the relation $\boldsymbol{x}_{i+1}^j = \boldsymbol{x}_i^{j+1}$. Assume that $|\boldsymbol{W}_{is} \cdot \boldsymbol{\delta}_{i-1}^j| \ll y_i^j$ and find in the lowest expansion order

$$\boldsymbol{\delta}_i^j \approx \frac{\partial f(\boldsymbol{y}_i^j)}{\partial \boldsymbol{y}_i^j} \cdot (\boldsymbol{W}_{is} \cdot \boldsymbol{\delta}_{i-1}^j),$$

and $\boldsymbol{M}_i^j = \partial f(\boldsymbol{y}_i^j)/\partial \boldsymbol{y}_i^j$ is a square matrix having the same dimensions as the matrix $\boldsymbol{W}_{is}$. Recalling that the nonlinear transformation f is actually a scalar function applied to all elements of its vector argument $\boldsymbol{y}_i^j$ one can write for the diagonal matrix $\boldsymbol{M}_i^j = \boldsymbol{D}[f'(\boldsymbol{y}_i^j)] \equiv \mathrm{diag}\{f'(\boldsymbol{y}_i^j)\}$. This leads to

$$\boldsymbol{\delta}_i^j \approx \boldsymbol{N}_i^j \cdot \boldsymbol{\delta}_{i-1}^j, \quad \boldsymbol{N}_i^j = \boldsymbol{D}[f'(\boldsymbol{y}_i^j)] \cdot \boldsymbol{W}_{is} = \boldsymbol{D}[f'(f^{-1}(\boldsymbol{s}_{i+1}^j))] \cdot \boldsymbol{W}_{is}. \tag{A3}$$

For basic RNN $f'(x) = \tanh' x = \mathrm{sech}^2 x$, leading to $f'(f^{-1}(x)) = 1 - x^2$, and thus

$$\boldsymbol{N}_i^j = \boldsymbol{D}[1 - \boldsymbol{s}_{i+1}^j \otimes \boldsymbol{s}_{i+1}^j] \cdot \boldsymbol{W}_{is}. \tag{A4}$$

Noting that due to (A1) with $f(x) = \tanh x$ we deduce that all elements of the vector $\boldsymbol{s}_i^j$ are in the range $\{-1, 1\}$ that implies $\boldsymbol{W}_{is}$ to be a majorant of $\boldsymbol{N}_i^j$ and thus the dynamics of the shifted difference is determined by the matrix $\boldsymbol{W}_{is}$. Consider a set of eigenvalues λ_k and (orthonormal) eigenvectors $\boldsymbol{e}_k$ of the matrix $\boldsymbol{W}_{is}$ satisfying $\boldsymbol{W}_{is} \cdot \boldsymbol{e}_k = \lambda_k \boldsymbol{e}_k$. For any two vectors $\boldsymbol{a}$, $\boldsymbol{b}$ in $\boldsymbol{b} = \boldsymbol{W}_{is} \cdot \boldsymbol{a}$ we have

$$\boldsymbol{b} = \sum_k \omega_{bk} \boldsymbol{e}_k = \sum_k \omega_{ak} \boldsymbol{W}_{is} \cdot \boldsymbol{e}_k = \sum_k \omega_{ak} \lambda_k \boldsymbol{e}_k. \tag{A5}$$

In case when all $|\lambda_k| < 1$ we find for the square of norms

$$b^2 = \sum_k \omega_{bk}^2 = \sum_k \omega_{ak}^2 |\lambda_k|^2 < \sum_k \omega_{ak}^2 = a^2,$$

and thus the transformation determined by $\boldsymbol{W}_{is}$ is contracting one. Computation of the spectrum of $\boldsymbol{W}_{is}$ for the trained basic RNN shows that the condition $|\lambda_k| < 1$ holds explaining the observed exponential decay of the shifted difference norm δ_i^j.

References

[1] H. Georgiou, S. Karagiorgou et al, Moving object analytics: survey on future location & trajectory prediction methods, Technical report, 2018, `arxiv:1807.04639 [cs.LG]`.

[2] P.R. Vlachas, W. Byeon et al, Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks, Proc. R. Soc. A 2017, **474**, 20170844.

[3] Q. Li, R.-C. Lin, A new approach for chaotic time series prediction using recurrent neural networks, Mathematical Problems in Engineering, 2016, **2016**, ID 3542898.

[4] R. Yu, S. Zheng, Y. Liu, Learning chaotic dynamics using tensor recurrent neural networks, Proceedings of theICML 17 Workshop on Deep Structured Prediction, Sydney, Australia, PMLR 70, 2017.

[5] S. Haykin (Ed.), *Kalman filtering and neural networks*, John Wiley, 2001.

[6] K. Yeo, Short note on the behavior of recurrent neural network for noisy dynamical system, 2019, `arxiv:1904.05158 [cs.NE]`.

[7] S. Hochreiter, J. Schmidhuber, Long-short term memory, Neural. Comput., 1997, **9**, 1735-1780.

[8] J. Chung, C. Gulcere, K.H. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural network for sequence modeling, 2014, `arxiv:1412:3555v1[cs.NE]`.

[9] A. Vaswani, N.Shazeeer et. al. Attention is all you need, 2017, `arxiv:1706.03762v5 [cs.CL]`.

[10] T.B. Brown, B. Mann et. al. Language models are few-shot learners, 2020, `arxiv:2005.14165v4 [cs.CL]`.