Physica D 470 (2024) 134371

Contents lists available at ScienceDirect

Physica D

journal homepage: www.elsevier.com/locate/physd

# Learning on predictions: Fusing training and autoregressive inference for long-term spatiotemporal forecasts

P.R. Vlachas [a,*], P. Koumoutsakos [a,b]

[a] Computational Science and Engineering Laboratory, ETH Zürich, Clausiusstrasse 33, Zürich CH-8092, Switzerland
[b] Computational Science and Engineering Laboratory, Harvard John A. Paulson School of Engineering and Applied Sciences, Harvard University, 29 Oxford Street, Cambridge, MA 02138, USA

ARTICLE INFO

Communicated by Boumediene Hamzi

*Keywords:*
Autoregressive forecasting
RNN
LSTM
BPTT
Exposure bias

ABSTRACT

Predictions of complex systems ranging from natural language processing to weather forecasting have benefited from advances in Recurrent Neural Networks (RNNs). RNNs are typically trained using techniques like Backpropagation Through Time (BPTT) to minimize one-step-ahead prediction loss. During testing, RNNs often operate in an auto-regressive mode, with the output of the network fed back into its input. However, this process can eventually result in exposure bias since the network has been trained to process "ground-truth" data rather than its own predictions. This inconsistency causes errors that compound over time, indicating that the distribution of data used for evaluating losses differs from the actual operating conditions encountered by the model during training. Inspired by the solution to this challenge in language processing networks we propose the Scheduled Autoregressive Truncated Backpropagation Through Time (BPTT-SA) algorithm for predicting complex dynamical systems using RNNs. We find that BPTT-SA effectively reduces iterative error propagation in Convolutional and Convolutional Autoencoder RNNs and demonstrates its capabilities in the long-term prediction of high-dimensional fluid flows.

## 1. Introduction

Deep learning (DL) methods have been instrumental to advances in a wide range of scientific disciplines from physics [1,2], and fluid dynamics [3,4], to climate modeling [5], and life sciences [6,7]. DL relies on neural network architectures, whose parameters are determined through optimization algorithms such as stochastic gradient descent. In turn, the critical computational bottleneck for DL is the efficient computation of gradients in error backpropagation (BP).

Recurrent neural networks (RNNs) are commonly used to handle sequential or temporal data as they efficiently take into account the sequential aspect of the tasks. The extension of the backpropagation (BP) algorithm to RNNs and temporal tasks is known as backpropagation-through-time (BPTT) [8–10]. BPTT has found widespread application in natural language processing [11,12], signal and image processing [13–15], speech recognition [16,17] and forecasting of complex dynamical systems [18–30]. It has been instrumental in solving complex temporal credit assignment problems [31,32]. In the computer vision community, video prediction has been a recent research focus [33–38], and BPTT is often utilized in a probabilistic context, such as in variational RNNs [38,39].

Recent studies have shown that RNNs can effectively model and forecast high-dimensional chaotic spatiotemporal dynamics [20,26,27,40–43]. Moreover, RNNs coupled with Convolutional Neural Networks (CNNs), [44–46] can be employed to model high dimensional spatiotemporal data, such as flow fields or images. The importance of long-term prediction of fluid flows is paramount for various practical cases from the prediction of extreme events [47], traffic management [48], surrogate modeling [45], typhoon alert systems, to climate and precipitation forecasting [44,49–51].

Most commonly, RNNs are trained using the Backpropagation Through Time (BPTT) algorithm in the teacher-forcing (BPTT-TF) mode [52], where the network is trained to minimize the error in one-step ahead predictions using sequences from the training dataset as input. Training RNNs is difficult due to vanishing and exploding gradients during backpropagation [53,54]. The recent study in [55] discusses the challenges of using BPTT-TF for simple RNN structures, specifically Elman RNNs when dealing with chaotic dynamics. This noteworthy research provides a theoretical understanding of the issue by associating the loss gradients observed during RNN training with the Lyapunov spectrum of the orbits, a concept pertinent to chaotic dynamics. However, the present work suggests that the weight gradients computed during training in this mode may be biased towards

* Corresponding author.
*E-mail addresses:* pvlachas@ethz.ch (P.R. Vlachas), petros@seas.harvard.edu (P. Koumoutsakos).

https://doi.org/10.1016/j.physd.2024.134371
Received 12 December 2023; Received in revised form 15 April 2024; Accepted 6 September 2024
Available online 16 September 2024

one-step-ahead predictions. The training loss is computed based on the probability distribution of the training data, which may not match the probability distribution of the testing data during the autoregressive testing phase, where the network uses its own predictions as input. This discrepancy is known as exposure bias [56], and it can negatively impact the generalization performance of the RNN.

To address this issue, alternative training methods such as scheduled sampling [57] or curriculum learning [58] can be employed to train the network in a way that better matches the distribution of the testing data. Curriculum learning strategies that quantify the complexity of the dataset using its entropy and gradually increase the dataset complexity during training have also been proposed for dynamical systems [59]. The network can better generalize to unseen data in this mode and produce more accurate predictions.

The primary objective of these works is to address the discrepancy in the teacher-forcing mode by incorporating techniques to replace, mask, or alter the ground-truth context. Several studies have attempted to overcome the limitations of BPTT-TF in the NLP domain. For example, some studies have employed deep auto-regressive models without hidden memory states, such as those discussed in [60], including models like PixelCNN [61], WaveNet [15], and models that use attention mechanisms [62,63]. These approaches improve the generalization performance of RNNs and overcome the issue of exposure bias.

The present approach for training autoregressive deep learning models for time series forecasting is based on the scheduled sampling method introduced in [57]. Backpropagation through time with scheduled sampling (BPTT-SS) is a training method used in training probabilistic RNNs, especially in sequence generation tasks like language modeling or machine translation. Probabilistic RNNs do not output a single state but a parametrization of a posterior distribution of the following states. The key idea behind BPTT-SS is to gradually transition the model from training on ground truth data to training on its own sampled predictions from the posterior distribution. At the beginning of the training, the model is fed with the ground truth data as input for the next step prediction. This is similar to standard supervised learning, where the model learns to predict the next item in a sequence based on the actual previous items. As training progresses, the method introduces samples from the posterior distribution (model's predictions) into the training process. Instead of always using the ground truth data from the previous item to predict the next one, the model occasionally uses its own sampled prediction from the previous step.

In our study, we utilize RNNs with deterministic rather than probabilistic outputs. Consequently, instead of generating the posterior distribution of the subsequent word or letter as in language processing, our RNNs produce a vector or field depicting the state's evolution. This introduces a significant distinction: the output tensor in our model allows for gradient flow, unlike the sample from the posterior in BPTT-SS. Therefore, our approach enables backpropagation through the predicted outputs, which are not based on a posterior distribution. We call this method Scheduled Autoregressive Backpropagation Through Time (BPTT-SA). It integrates an auxiliary loss to compensate for autoregressive forecasting errors and modifies the BPTT computational graph to enhance gradient computation.

Similar to BPTT-SS, the training process starts with a standard one-step ahead of prediction loss and gradually switches to autoregressive loss as training progresses. However, during gradient propagation, BPTT-SS uses samples from the predictive posterior distribution, whereas our proposed method, BPTT-SA, also backpropagates through these predictions. BPTT-SA aims to address exposure bias in RNNs by explicitly accounting for the dissimilarity between the probability distributions of training and testing data. This enables the proposed approach to enhance the model's generalization performance in autoregressive forecasting. In this work, we evaluate the effectiveness of BPTT-SA in deterministic spatiotemporal prediction using RNNs and compare it to standard BPTT and the scheduled sampling approach of [57].

We note that BPTT-TF is biased towards short-term forecasting as it minimizes one-step-ahead prediction error. In turn, the autoregressive loss prioritizes long-term error. Balancing these two objectives is challenging, even in the case of linear prediction models [64]. We conducted a comparative study on the Mackey glass systems and the Darwin sea level temperature time-series dataset and found that the benefits of BPTT-SA are not significant for low-dimensional time series prediction. However, in the solution of the Navier–Stokes equations for flow past a circular cylinder, BPTT-SA can reconcile the short-term accuracy and long-term accuracy objectives more effectively compared to the scheduled sampling approach of [57], without incurring extra training cost. Additionally, BPTT-SA helps alleviate the propagation of errors and enhances long-term prediction.

We remark that a recent study by Teutsch et al. [65] explored training algorithms for RNNs, focusing on similar curriculum learning methods. In addition, a version of BPTT-TF that uses sparsely applied teacher-forcing signals is employed for modeling EEG signals [66,67], but it has not been thoroughly discussed or analyzed in the context of spatiotemporal modeling. The research conducted in these studies only extends to the usage of these techniques on chaotic systems of low order. Our work expands on these findings by demonstrating that the benefits of employing curriculum learning and sparse teacher-forcing signals for Recurrent Neural Networks (RNNs) in prediction tasks are even more evident in dynamic systems with high dimensions.

## 2. Models and methods

Recurrent neural networks (RNNs) are deep learning architectures that handle sequential data. They process an input stream $x_0, \dots, x_T$ sequentially, where each element in the stream is a state $x \in \mathbb{R}^{d_x}$. RNNs have an internal hidden state that encodes information about the history of the input stream. The functional form of the RNN is described by:

$$h_t = f(x_t, h_{t-1}; w_h) \tag{1}$$

$$o_t = g(h_t; w_o) \tag{2}$$

where $f(x_t, h_{t-1}; w_h)$ is the recurrent (hidden-to-hidden) mapping, and $g(h_t; w_o)$ is the output (hidden-to-output) mapping $w_h$ and $w_o$ are trainable weights of the mappings. Gated Recurrent Units [68] and Long Short-Term Memory units [69] are possible implementations of RNN mappings. They both can be framed under this unifying lens.

In multiple tasks, RNNs are used as regressors for forecasting the evolution of the state $x_t$. In such cases, the output is a prediction of the state at the next timestep $x_{t+1}$, i.e.

$$o_t = \widetilde{x}_{t+1} = g(h_t; w_o) = w_o h_t, \tag{3}$$

where $w_o \in \mathbb{R}^{d_x \times d_x}$. The weights are optimized to minimize the prediction loss $\mathcal{L}(x_{t+1}, o_t)$, e.g., the mean squared error:

$$\mathcal{L}(x_{t+1}, o_t) = |x_{t+1} - o_t|_2^2 = |x_{t+1} - \widetilde{x}_{t+1}|_2^2, \tag{4}$$

where $|\cdot|$ is the L2-Norm. The loss measures the difference between the RNN prediction $o_t$ and the target value $x_{t+1}$.

### 2.1. Backpropagation through time with teacher-forcing

The Backpropagation Through Time with Teacher-Forcing (BPTT-TF) seeks to predict $t+1$ given a stream of ground-truth data $\{x_0, \dots, x_t, x_{t+1}\}$, that is provided as input to the network. The network is unrolled for $t+1$ timesteps during the forward pass, applying the Eq. (2) iteratively. In practice, due to memory and computational limitations, the input data stream for predicting the output at time $t+1$ is truncated to the last $L$ timesteps (sequence length), i.e., $\{x_{t-L+1}, \dots, x_t\}$. A schematic view of the computational graph of BPTT-TF and the backward flow of the gradient is shown in Fig. 1.

The network's output $o_t$ (and thus the loss defined in Eq. (4)) is a function of the initialization of the hidden state $h_0$, the input stream, and the RNN weights $\{w_h, w_o\}$, i.e.,

$$o_t = \mathrm{TF}\Big( \underbrace{h_0,}_{\text{initial hidden state}} \underbrace{x_0, \dots, x_t}_{\text{input stream}} ; \underbrace{w_h, w_o}_{\text{weights}} \Big). \tag{5}$$

The loss function is defined in Eq. (4). Following [70], the gradients of the loss with respect to the network's weights computed with backpropagation are given by:

$$\frac{\partial \mathcal{L}(x_{t+1}, o_t)}{\partial w_o} = \frac{\partial \mathcal{L}(x_{t+1}, o_t)}{\partial o_t} \frac{\partial g(h_t; w_o)}{\partial w_o}, \tag{6}$$

$$\frac{\partial \mathcal{L}(x_{t+1}, o_t)}{\partial w_h} = \frac{\partial \mathcal{L}(x_{t+1}, o_t)}{\partial o_t} \frac{\partial g(h_t; w_o)}{\partial h_t} \frac{\partial h_t}{\partial w_h}. \tag{7}$$

The term $\partial \mathcal{L}(x_{t+1}, o_t)/\partial o_t$ is evaluated by Eq. (4) and $\partial g(h_t; w_o)/\partial w_o$ by Eq. (3). Evaluation of the term $\partial h_t / \partial w_h$ involves a recurrence, as $h_t$ depends on $h_{t-1}$ and $w_h$, while $h_{t-1}$ also depends on $w_h$. The gradient can be evaluated using the chain rule

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{w_h}. \tag{8}$$

For the evaluation of this gradient we utilize the following lemma, whose proof can be found in Appendix A.

**Lemma 1.** *Assume that there are three sequences $a_t, b_t$ and $c_t$, with $a_0 = 0$ and $a_t = b_t + c_t a_{t-1}$ for $t \in 1, 2, \dots, T$. For $t > 1$ it holds that:*

$$a_t = b_t + \sum_{i=1}^{t-1} \Big( \prod_{j=i+1}^{t} c_j \Big) b_i. \tag{9}$$

By setting:

$$a_t = \frac{\partial h_t}{\partial w_h}, \quad b_t = \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial w_h}, \quad c_t^{\mathrm{TF}} = c_t = \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial h_{t-1}}, \tag{10}$$

the three series satisfy the requirements of the lemma, namely $a_t = b_t + c_t^{\mathrm{TF}} a_{t-1}$ as per Eq. (8), with $a_0 = 0$. As a consequence, by substituting Eq. (10) into Eq. (9), we end up with:

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \Bigg( \prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}; w_h)}{\partial h_{j-1}} \Bigg) \frac{\partial f(x_i, h_{i-1}; w_h)}{\partial w_h}. \tag{11}$$

Note that the gradient in Eq. (11) is relevant when the input sequence in the forward pass consists of ground-truth data. A common problem encountered in practice during RNN training is vanishing and exploding gradients during backpropagation [53,54]. The gradient Eq. (11) entails the product of $c_t^{\mathrm{TF}}$s (defined in Eq. (10)). Successful training with BPTT (i.e., capturing long-term dependencies, loss reduction, informative gradients, and non-oscillatory loss behavior) depends on keeping this gradient at a reasonable norm.

In the truncated BPTT-TF, the gradient Eq. (11) is not computed over the whole sequence. The forward propagation is performed over a subset of the input stream consisting of the last $L$ steps, and the backpropagation is truncated, ignoring the history prior to these $L$ timesteps. The hidden state $h_0 = h_{t-L+1}$ at the point where the gradient flow is truncated can be set from the previous batch (state-full RNN) or set to zero in state-less RNNs. Here, we consider the state-full RNN case.

### 2.2. Backpropagation through time with autoregression

The previous section (Section 2.1) explained that when a network is trained with teacher-forcing, it learns to predict the state evolution at a future timestep for a specific lead time. However, in practical applications, the network must often provide forecasts for multiple lead times or forecast the future evolution of a data stream. Training multiple models for different lead times is computationally expensive and not scalable, and it is preferable to use a trained model for future data stream trajectory forecasts. Autoregressive inference can achieve this by feeding the output of the RNN back into the input, iteratively forecasting the data stream's evolution. However, using teacher-forcing in this case poses a significant problem, as the network is not trained to generate predictions based on its own outputs at the input, but rather on ground-truth data. The network outputs might also follow a different distribution than the ground-truth data due to imperfect training.

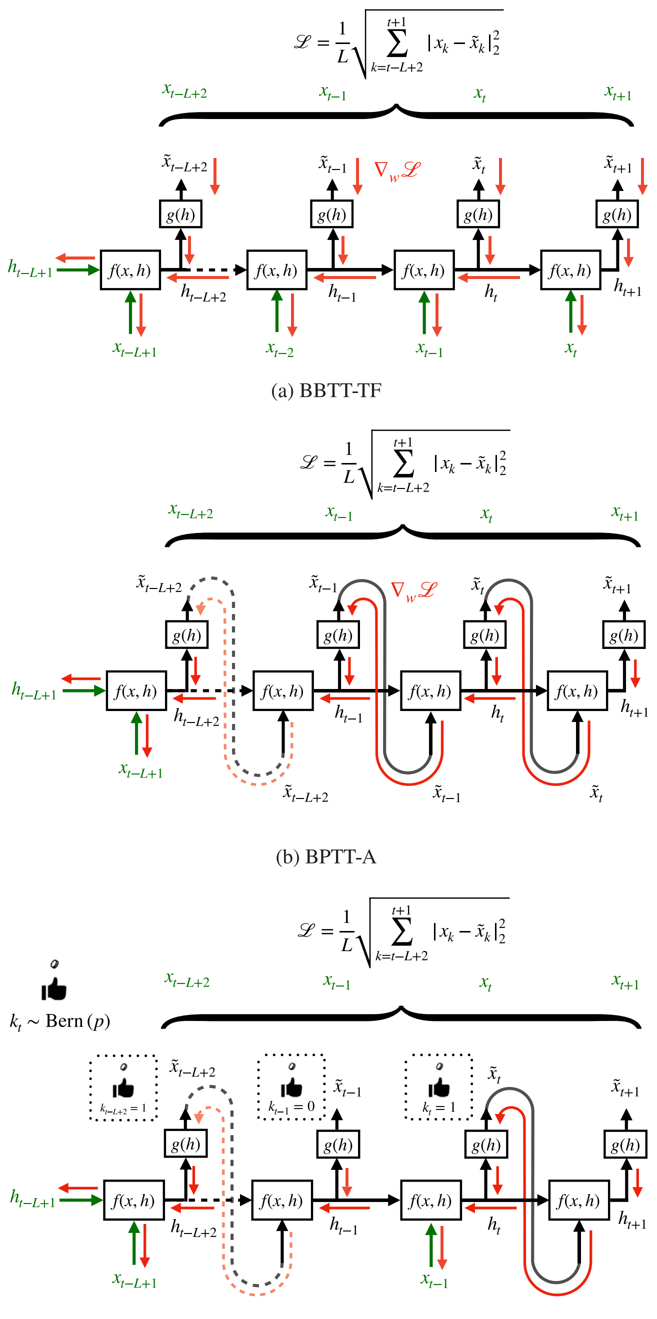


**Fig. 1.** (a) Illustration of the forward pass and backward gradient flow of Backpropagation Through Time with Teacher-Forcing (BPTT-TF), with green indicating the data (input and target). Black arrows represent the forward pass of the network, while the backward gradient pass is depicted in red. (b) In Autoregressive BPTT (BPTT-A), the network's output is propagated through the network, only providing input data at the first time step. (c) In Scheduled Autoregressive BPTT (BPTT-SA), the behavior depends on the autoregressive probability $p$ parametrizing a Bernoulli distribution.

We address these challenges by proposing an alternative training method called Autoregressive BPTT and derive the associated gradient. In Autoregressive BPTT, in order to compute the output at timestep $t$, the network iteratively propagates its own predictions, beginning with an initial hidden state $h_0$ and the first input data state of the stream $x_0$. The output of the network is determined by the weights of the network and its initial state, $h_0$ and $x_0$, i.e.,

$$o_t = \mathrm{IF}\big(\underbrace{h_0,}_{\text{initial state}} \underbrace{x_0}_{\text{initial input}} ; \underbrace{w_h, w_o}_{\text{weights}}\big). \tag{12}$$

Note the difference compared to the equation of the output in teacher-forcing in Eq. (5). The gradient of the loss with respect to the network's parameters in this case is different. We start by repeating the gradients from Eq. (7):

$$\frac{\partial \mathcal{L}(x_{t+1}, o_t)}{\partial w_o} = \frac{\partial \mathcal{L}(x_{t+1}, o_t)}{\partial o_t} \frac{\partial g(h_t; w_o)}{\partial w_o}$$
$$\frac{\partial \mathcal{L}(x_{t+1}, o_t)}{\partial w_h} = \frac{\partial \mathcal{L}(x_{t+1}, o_t)}{\partial o_t} \frac{\partial g(h_t; w_o)}{\partial h_t} \frac{\partial h_t}{\partial w_h}$$

In the following, we treat the input $x_t$ as a function of the previous hidden state $x_t = g(h_{t-1})$ due to autoregression. This implies:

$$\begin{aligned}\frac{\partial h_t}{\partial w_h} &= \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial x_t} \underbrace{\frac{\partial g(h_{t-1})}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h}}_{\frac{\partial x_t}{\partial w_h}} \\ &\quad + \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h} \implies \\ &= \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial w_h} \\ &\quad + \left( \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial x_t} \frac{\partial g(h_{t-1})}{\partial h_{t-1}} + \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial h_{t-1}} \right) \frac{\partial h_{t-1}}{\partial w_h}\end{aligned} \tag{13}$$

To evaluate the recurrence relation and evaluate the gradient in Eq. (13), we use again Lemma 1. By setting

$$\begin{aligned}a_t &= \frac{\partial h_t}{\partial w_h}, \quad b_t = \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial w_h}, \\ c_t^{\mathrm{AR}} &= c_t = \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial x_t} \frac{\partial g(h_{t-1})}{\partial h_{t-1}} + \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial h_{t-1}}\end{aligned} \tag{14}$$

the three series satisfy the requirements of the lemma, namely $a_t = b_t + c_t^{\mathrm{AR}} a_{t-1}$ as per Eq. (13), with $a_0 = 0$. As a consequence, by substituting Eq. (14) into Eq. (9), we end up with:

$$\begin{aligned}\frac{\partial h_t}{\partial w_h} &= \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial w_h} \\ &\quad + \sum_{i=1}^{t-1} \left( \prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}; w_h)}{\partial x_j} \frac{\partial g(h_{j-1})}{\partial h_{j-1}} + \frac{\partial f(x_j, h_{j-1}; w_h)}{\partial h_{j-1}} \right) \\ &\quad \times \frac{\partial f(x_i, h_{i-1}; w_h)}{\partial w_h}\end{aligned} \tag{15}$$

In contrast to the gradient in the teacher-forcing case (Eq. (11)), the gradient in the autoregressive case contains a product that also involves the hidden-to-output mapping, i.e., $\partial g(h_{j-1})/\partial h_{j-1}$. Consequently, the hidden-to-output mapping is also regularized, assuming no vanishing or exploding gradients. Previous studies on forecasting high dimensional dynamical systems have reported that RNNs trained with teacher-forcing tend to produce unrealistic patterns and diverge from the underlying attractors in autoregressive testing [71]. We argue that one of the main reasons for this degeneracy is that the teacher-forcing does not regularize the hidden-to-output mapping, as it is not involved in the product (recursive time unrolling) in the gradient. Consequently, although predictions are accurate in the short term, the network's output weights are not regularized for the iterative propagation of the output, causing divergence of the predictions. This effect is more prominent in long-term spatiotemporal forecasting of high-dimensional dynamical systems, where the hidden-to-output mapping can be a large Convolutional Neural Network (CNN) with thousands of parameters, as opposed to low-order systems. A schematic view of Autoregressive BPTT (BPTT-A) is given in Fig. 1(b).

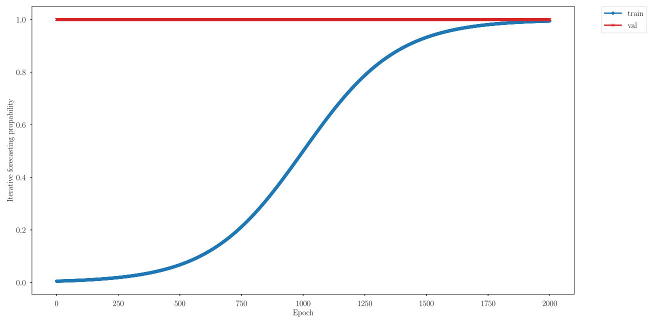


**Fig. 2.** Schedule of iterative forecasting probability $p$.

## 2.3. *Truncated backpropagation through time with scheduled autoregression*

We propose a scheduled autoregressive approach inspired by [57]. Before training an RNN, its weights are initialized randomly, and the network cannot generate accurate short-term predictions. Propagating these imprecise predictions iteratively results in a gradual buildup of errors. Although the RNN improves with training, its short-term predictions remain inaccurate in the initial training epochs. Hence, utilizing the autoregressive gradient during the initial training period is inaccurate, even on short-term predictions.

In Scheduled Autoregressive BPTT (BPTT-SA), the selection of the propagation type at each timestep depends on a sample $k_t \in \{0, 1\}$ from a Bernoulli distribution, parametrized by the iterative forecasting probability $p$, i.e., $k_t \sim \mathrm{Bern}(p)$. $k_t$ is one with probability $p$ and zero with probability $1 - p$ (coin-flip). At each timestep $t$, we sample a different $k_t$ and decide on BPTT-TF if $k_t = 0$ or BPTT-A if $k_t = 1$. Following the argumentation in Section 2.1 and Section 2.2, the gradient $\frac{\partial h_t}{\partial w_h}$ is evaluated as:

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}; w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left( \prod_{j=i+1}^{t} \left(c_j^{\mathrm{TF}}\right)^{(1-k_j)} \left(c_j^{\mathrm{AR}}\right)^{k_j} \right) \frac{\partial f(x_i, h_{i-1}; w_h)}{\partial w_h}. \tag{16}$$

Note that for $p = 0$, $k_j \sim \mathrm{Bern}(0) = 0$, the product $\left(c_j^{\mathrm{TF}}\right)^{(1-k_j)} \left(c_j^{\mathrm{AR}}\right)^{k_j}$ in Eq. (16) is equal to $c_j^{\mathrm{TF}}$ and the gradient in Eq. (16) evaluates to the BPTT-TF gradient expressed in Eq. (11). In contrast, for $p = 1$, the aforementioned product is equal to $c_j^{\mathrm{AR}}$ and the gradient evaluates to the BPTT-A gradient expressed in Eq. (15).

During training, the parametrization of the Bernoulli distribution, i.e., $p$, follows an inverse sigmoid schedule for the training loss. In the first epoch $p = 0$, samples from $k_t \sim \mathrm{Bern}(0)$ are all 0, and the model is trained with BPTT-TF. As training progresses, the parameter $p$ is gradually annealed until it reaches $p \to 1$, more and more samples $k_t \sim \mathrm{Bern}(p)$ are 1, predictions are propagated in the forward pass, and the gradient is also backpropagated through these predictions (BPTT-SA). Towards the final training epochs when $p = 1$, all samples $k_t \sim \mathrm{Bern}(p)$ are 1, and the network is trained with the gradient of BPTT-A. The validation loss is computed for $p = 1$ (equivalent to BPTT-A). This ensures that the validation loss according to which we pick the optimal model is the autoregressive loss. The schedule is depicted in Fig. 2 for a training procedure of 2000 epochs in total.

## 2.4. *Difference between scheduled autoregression and scheduled sampling*

The proposed BPTT-SA method is inspired by the work of Bengio et al. [57]. We remark on an essential distinction in the context of neural networks and differentiable systems. In [57], the authors indeed use a form of the recurrence $x_{t+1} = o_t$. The output $o_t$ of their network is a parametrization of a probability distribution, and $x_{t+1}$ is a sample from this distribution. However, during backpropagation, they are creating a non-differentiable point in the computation graph of the model, which has significant implications for learning.

In order to understand the difference, we consider a simple example, i.e., a linear RNN described by:

$$h_t = w_x x_t + w_h h_{t-1},$$
$$o_t = w_o h_t,$$

with initial conditions:

$$h_0 = 0,$$
$$x_1 = d_1,$$
$$x_t = o_{t-1} \text{ for all } t > 1.$$

The gradients of BPTT-SS, computed in Appendix B, are:

$$\begin{aligned}\frac{\partial o_2}{\partial w_o} &= w_o w_x^2 d_1 + w_h w_x d_1,\\ \frac{\partial o_3}{\partial w_o} &= w_o^2 w_x^3 d_1 + 2 w_o w_h w_x^2 d_1 + w_h^2 w_x d_1,\end{aligned} \tag{17}$$

The gradients of BPTT-SA, computed in Appendix C, are:

$$\begin{aligned}\frac{\partial o_2}{\partial w_o} &= 2 w_o w_x^2 d_1 + w_h w_x d_1, .\\ \frac{\partial o_3}{\partial w_o} &= 3 w_o^2 w_x^3 d_1 + 4 w_o w_h w_x^2 d_1 + w_h^2 w_x d_1.\end{aligned} \tag{18}$$

We notice that the gradients contain terms with similar dependencies on the weights, but the factors are different. The difference in the functional form between BPTT-SS and BPTT-SA, in the context of a linear RNN, is evident through the disparate scaling of the terms, as illustrated in Eqs. (17) and (18). Nonetheless, in the realm of more sophisticated RNN architectures, entirely distinct terms may emerge. The simplicity in this case stems from the linear output mapping $x_t = o_{t-1} = w_o h_{t-1}$, and the linear hidden-to-hidden connections of the RNN.

To illustrate that the differences extend beyond merely scaling factors, we undertake the gradient computation for a slightly more nuanced scenario: an RNN characterized by linear hidden-to-hidden dynamics and a non-linear hidden-to-output mapping, employing a tanh function for output activation. The gradients are provided in Appendix D and Appendix E. To highlight the contrasting gradients in this context, it is unnecessary to evaluate $\partial o_3/\partial w_o$. The emergence of a completely new term for $\partial o_2/\partial w_o$ already signifies this divergence. The gradient $\partial o_2/\partial w_o$ has a more complicated form, with extra terms appearing only on BPTT-SA:

$$\frac{\partial o_2}{\partial w_o} = \frac{w_x\left(d_1 w_h + \frac{d_1 w_o w_x}{\cosh^2(d_1 w_o w_x)} + \tanh(d_1 w_o w_x)\right)}{\cosh^2\left(w_o w_x\left(d_1 w_h + \tanh(d_1 w_o w_x)\right)\right)} \tag{19}$$

The difference is that with gradient backpropagation on $x_t$, the dependency $x_t = o_{t+1}$ forms a closed-loop system where the output at one step becomes the input at the next. When backpropagating through this system, gradients flow through the $o_{t-1}$ to $x_t$ connection. Consequently, the model will consider the impact of the weights on previous outputs $o_{t-1}$ when adjusting the weights due to $x_t$. Essentially, the model learns to correct its future inputs by understanding its past errors in output.

If the gradient on $x_t$ is disabled, $o_{t-1}$ is used as input for the next timestep, but the model ignores how changes to the weights affect the value of $o_{t-1}$. This creates a disconnect between the model's output and the subsequent input in the learning process is created. The model still generates $x_t$ from $o_{t-1}$ during the forward pass but does not fully account for how its updates might improve or worsen this generation process. The gradient-disabled approach might hinder the model's ability to learn from and correct its mistakes over sequences. If the model makes an error at time $o_{t-1}$, this error directly influences the input at time $t$ without giving the model a chance to correct the causal error. This leads to error accumulation across the sequence, potentially making learning less stable.

## 3. Results

The proposed BPTT-SA is compared against the standard BPTT-TF, and the method proposed in [57] (here denoted as BPTT-SS) in a number of benchmark problems. All methods are implemented in Pytorch [72] and ported to a single Nvidia Tesla P100 GPU. In all experiments, the models are trained with the Adam [73] optimizer, and we employ validation-based early stopping to cope with overfitting.

### 3.1. Mackey-Glass

In this section, we demonstrate the effectiveness of BPTT-SA in forecasting chaotic time series from the Mackey-Glass (MG) equations. This is a challenging benchmark problem due to its chaotic nature [74,75]. The time-series is generated by the delay differential equation

$$\frac{dx}{dt} = \frac{\alpha\, x(t-\tau)}{1 + x^n(t-\tau)} - \beta x(t). \tag{20}$$

We consider the parameter setting $\alpha = 0.2$, $\beta = 0.1$, $c = 10$, and $\tau = 17$.

The maximum Lyapunov exponent, calculated with the method of [26] is $\Lambda_1 \approx 8.9 \cdot 10^{-3}$, leading to a Lyapunov time of $T^{\Lambda_1} = 112$ time units. We integrate Eq. (20) with a fourth-order Runge–Kutta scheme with $\delta t = 0.1$ up to $T = 2 \cdot 10^5$. The data are subsampled after integration to $\Delta t = 1.0$. 32 sequences of 1120 timesteps each (10 Lyapunov times) are generated for training. A data set of the same size is considered for validation. The remaining data are considered for testing. In order to test the proposed algorithm in the autoregressive setting, 100 initial conditions are randomly sampled from the test data, and the networks are asked to forecast the next 896 steps, which amounts to approximately 8 Lyapunov times (after an initial warm-up period of 20 timesteps). As comparison metrics, we consider the Root Mean Square Error (RMSE) and the error on the power spectrum (frequency content). Moreover, we consider two different noise levels on the data, a signal-to-noise ratio of SNR = 60, and a disturbed case of SNR = 10.

The results are illustrated in Fig. 3. In the case of low-level noise (SNR = 60), all methods show approximately the same performance in terms of the RMSE. BPTT-SA shows slightly better performance on average and smaller variations between the different seeds (increased robustness) on the power spectrum error. However, the differences are small. In the SNR = 10 noise level, all three methods exhibit similar errors.

### 3.2. Darwin sea level pressure dataset

In this study, we conduct an evaluation of the long-term predictability of the methods using an open-source dataset that is widely utilized as a benchmark for time-series prediction techniques. The dataset reports the monthly average sea level pressure recorded at Darwin between 1882 and 1998 [76] and consists of 1400 samples. The first 600 samples are used for training and the next 400 for validation. The long-term forecasting accuracy of the methods is evaluated on 32 initial conditions randomly sampled from the test data. The RNNs forecast up to a prediction horizon of 100 timesteps after an initial warm-up period of 50 timesteps. The results are illustrated in Fig. 4. We observe that neither variant, BPTT-SA or BPTT-SS, offers any improvements in RMSE or the power spectrum error.

### 3.3. Navier–Stokes equations: Flow past a circular cylinder at Re=200

We consider long-term forecasting of the dynamics of the incompressible Navier–Stokes equations governing the flow past a circular

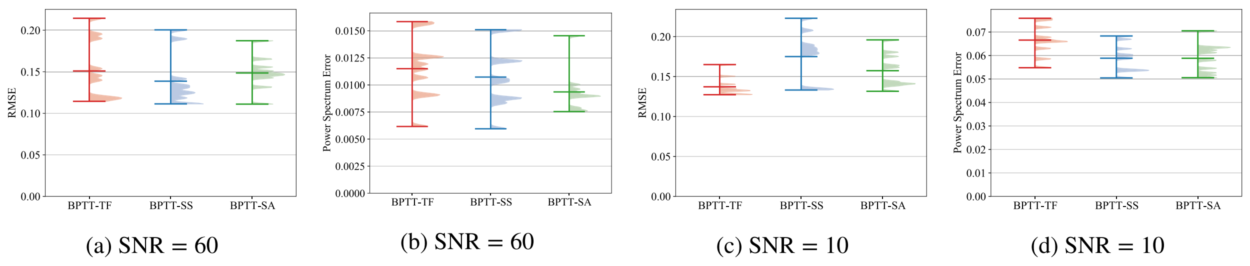


**Fig. 3.** Evaluation of the performance of the training methods in forecasting the long-term dynamics of the Mackey-Glass time-series. A prediction horizon of 896 time steps is considered, and results are averaged over 100 initial conditions in the test data.

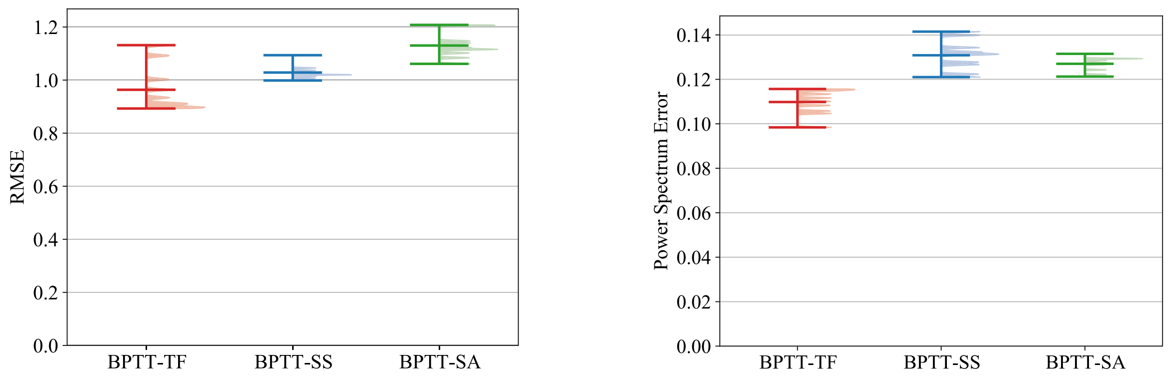


**Fig. 4.** Results on the Darwin sea level pressure dataset.

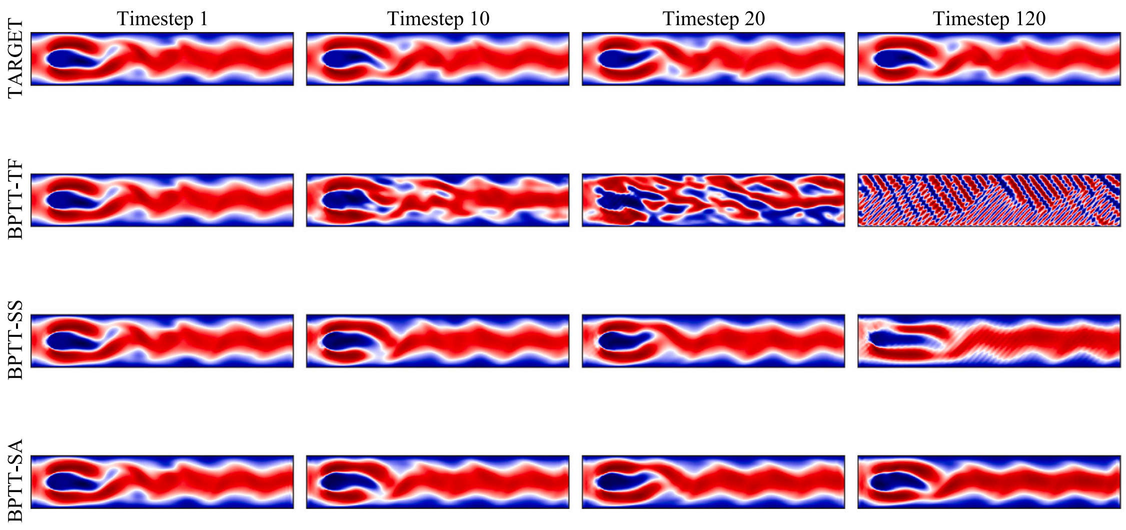


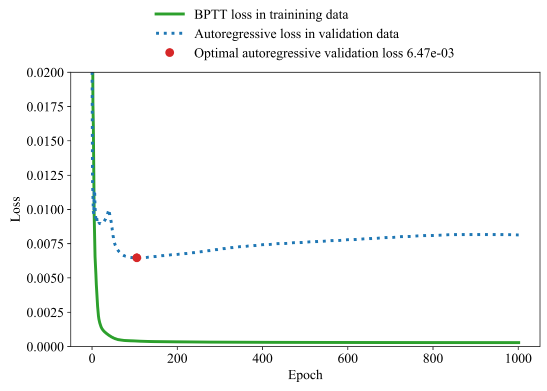
**Fig. 5.** Prediction samples in different timesteps in the autoregressive testing mode of ConvRNN networks trained with different methods in the Navier–Stokes dataset.

cylinder in a channel at Reynolds number $Re = 200$ in two dimensions. The flow exhibits a periodic vortex street, making long-term motion prediction possible. Moreover, the dimensionality of the intrinsic dynamics (effective degrees of freedom) of the motion is low [46]. This implies that snapshots of the flow can be mapped to a reduced-order latent space, representing the manifold of the effective dynamics [46].

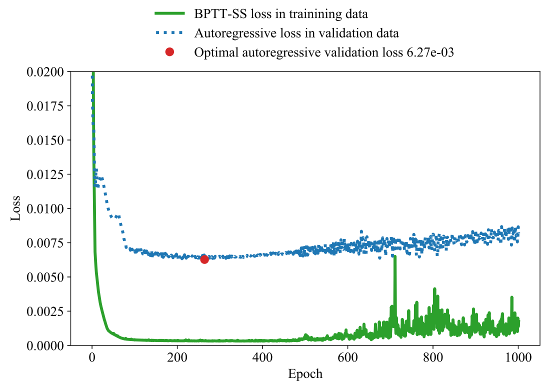
We evaluate the effectiveness of BPTT-SA in two types of recurrent neural networks: Convolutional RNNs (ConvRNN) and Convolutional Autoencoder RNNs (CNN-RNNs). The latter first identifies a latent reduced-order representation encoding the fluid flow's intrinsic dimensionality and learns the latent space's temporal dynamics. In contrast, ConvRNNs are replacing the operations on the RNN cell with convolutions while keeping the gating mechanisms [44,49] and do not require low dimensional intrinsic dynamics.

The flow is simulated with a Finite Element (FEM) solver [77] with a timestep of 0.001. The velocity $u \in \mathbb{R}^2$ and pressure $p \in \mathbb{R}$ values were extracted in a uniform grid of $160 \times 32$, and data are subsampled to $\Delta t = 0.01$. For plotting purposes, we plot the Frobenius norm of the velocity $u = \sqrt{u_x^2 + u_y^2}$. For more information on the geometry and simulation details, refer to [78]. The total simulation time is $T = 24$. The first 200 timesteps are used for training, the next 200 for validation, and the next 2000 for testing. The long-term iterative prediction performance of the methods is evaluated on the prediction of 1000 timesteps starting from 10 initial conditions randomly sampled from the test data. For more information about the hyperparameter tuning, refer to Appendix H.

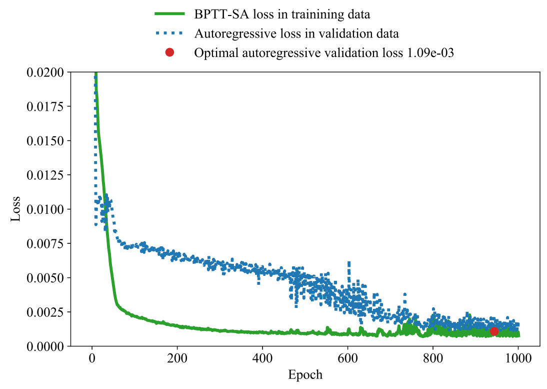
The evolution of the training and validation error in the CNN-RNN training on the Navier–Stokes dataset is given in Fig. 6. In BPTT-TF



(a) CNN-RNN trained with BPTT



(b) CNN-RNN trained with BPTT-SS



(c) CNN-RNN trained with BPTT-SA

**Fig. 6.** Evolution of the training and validation losses in CNN-RNN training in the Navier–Stokes dataset.

the training error is decreasing, but it does not capture the long-term autoregressive prediction error. For this reason, the autoregressive validation error is increasing, as the model is overfitting in the one-step-ahead prediction error. In BPTT-SS, the training error is encoding the autoregressive loss due to the scheduled sampling approach. However, as the method does not backpropagate the gradients, training is challenging as the iterative forecasting probability $p$ increases, and the training error is not reduced. For this reason, the validation error also remains high. In contrast, the autoregressive validation error decreases in the model trained with BPTT-SA, demonstrating that the training loss and the gradient capture and successfully encode the objective of long-term forecasting. Similar behavior is observed in Fig. 7 for training ConvRNN models.

In the autoregressive testing, we consider two comparison metrics, i.e., the RMSE error (the smaller, the better) and the structural similarity index measure (SSIM) [79] (the higher, the better). The performance of the CNN-RNN models is illustrated in Figs. 8(a) and 8(b). Four random seeds are considered to evaluate the robustness of the training algorithms. BPTT-SA leads, on average, to a drastic reduction of the RMSE and an increase in the SSIM. The same holds for ConvRNNs models as depicted in Figs. 8(c) and 8(d). CNN-RNNs exhibit lower errors in both metrics than ConvRNNs as they consider the reduced-order nature of the effective dynamics and predict on a low-dimensional latent space.

In Fig. 9, we plot the evolution of the RMSE and the SSIM errors in time. We observe that BPTT-SA alleviates the error propagation and leads to more accurate long-term predictions in both metrics. In Fig. 5, we plot samples from the autoregressive testing phase for the ConvRNN models. We observe that models trained with BPTT and BPTT-SS lead to unphysical predictions after some timesteps. At lead time $T = 120$, only BPTT-SA captures the flow characteristics.

## 4. Discussion

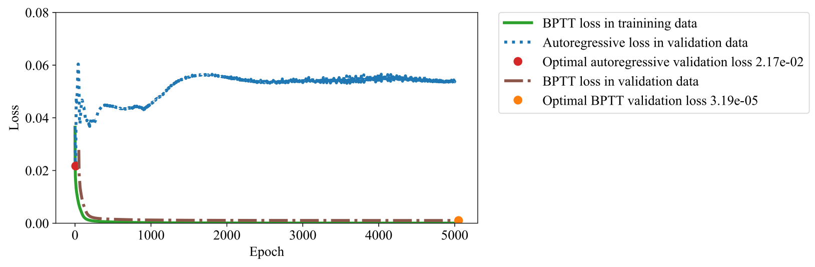
BPTT-SA is particularly useful in scenarios where creating models for long-term forecasting is necessary, but creating various models for different lead times is not possible or expensive. The BPTT-SA method applies to any recurrent architecture without additional training time or memory cost. BPTT-SA has many potential applications, including improving the long-term prediction capabilities of data-driven surrogate/reduced-order models of dynamical systems, Computational Fluid Dynamic (CFD), or Finite Element (FEM) codes, and environment dynamics models for model-based Reinforcement Learning [80]. Additionally, the method can be used to fine-tune any recurrent architecture to achieve state-of-the-art results in long-term prediction across various applications, as demonstrated by the recent work of [81]. However, in the case of low-dimensional time series, the results of this study indicate that the merits of BPTT-SA regarding long-term forecasting performance are marginal.

Moreover, as BPTT-SA is learning on previous predictions, backpropagating through the input $x_t = o_{t-1}$, is best suited for autonomous dynamical systems. In certain use cases, mainly where $x_t$ is determined by external factors (external forcing, user input, environmental factors), it might make sense to turn off the gradient through $x_t$ and use BPTT-SS because the model truly has no control over these values. However, in a closed-loop system where the model's outputs become its inputs, turning off this gradient can limit the model's potential for understanding and improving its sequential decision-making process.

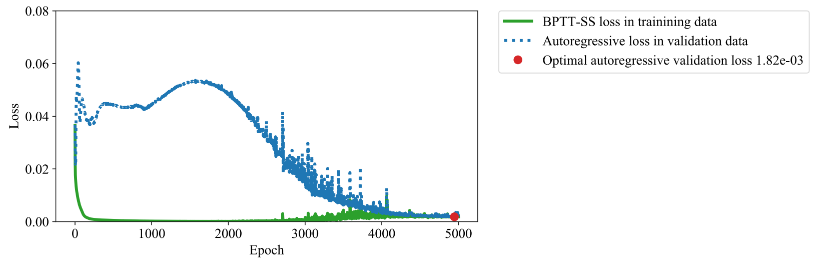
## 5. Summary

We introduce the scheduled autoregressive BPTT (BPTT-SA) method to address the exposure bias in RNNs during iterative forecasting. BPTT-SA is founded on the work of Bengio et al. [57] (here called BPTT-SS) and augments their method with backpropagation through the input $x_t$. This algorithm allows RNNs to learn on their own previous predictions, a feature missing in this previous work. We compare the performance of BPTT-SA to standard BPTT and BPTT-SS in low dimensional time-series problems and the Navier–Stokes flow past a cylinder, taking into account RNNs, convolutional encoder–decoder RNNs (CNN-RNNs), and RNNs with convolutional connections (ConvRNNs).

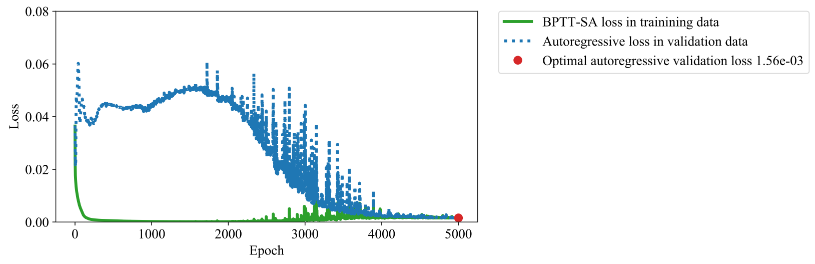
Our results demonstrate that BPTT-SA can effectively reduce errors in long-term, high-dimensional spatiotemporal prediction in ConvRNNs and CNN-RNNs for the Navier–Stokes flow without additional training costs. Future research will include the evaluation of the method in climate [50] and fluid flow [45] datasets, exploration of alternative sampling schedules, and the study of more sophisticated sampling mechanisms, i.e., importance sampling based on the prediction error.



(a) ConvRNN trained with BPTT

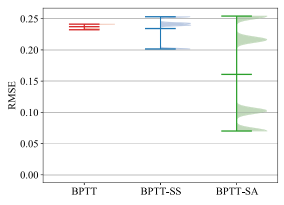


(b) ConvRNN trained with BPTT-SS

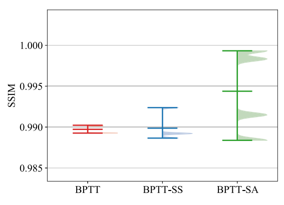


(c) ConvRNN trained with BPTT-SA

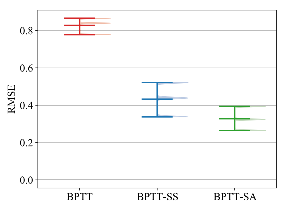
**Fig. 7.** Evolution of the training and validation losses in ConvRNN training in the Navier-Stokes dataset.



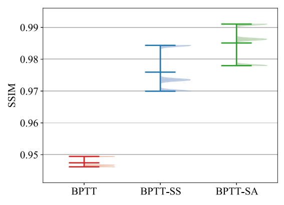
(a) CNN-RNN



(b) CNN-RNN

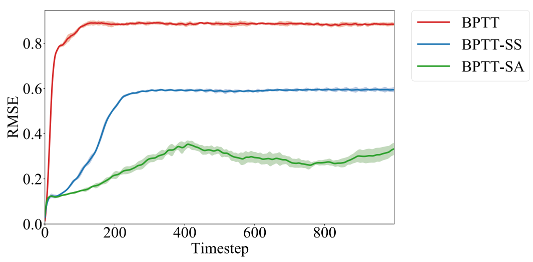


(c) Conv-RNN

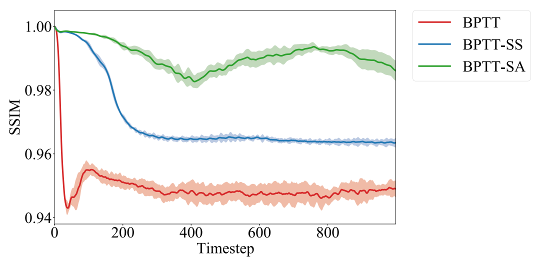


(d) Conv-RNN

**Fig. 8.** Evaluation of the performance of Scheduled Autoregressive BPTT (BPTT-SA) in long-term prediction on the Navier–Stokes flow past a cylinder for two model types, CNN-RNNs and ConvRNNs. The better prediction capability of a model is demonstrated by lower RMSE and higher SSIM scores.



(a) RMSE in ConvRNNs



(b) SSIM in ConvRNNs

**Fig. 9.** The evolution of the SSIM and RMSE errors in time in autoregressive long-term prediction in the test data on the Navier–Stokes flow past a cylinder for ConvRNNs. The better prediction capability of a model is demonstrated by lower RMSE and higher SSIM scores.

## CRediT authorship contribution statement

**P.R. Vlachas:** Writing – review & editing, Writing – original draft, Visualization, Software, Resources, Methodology, Investigation, Data curation, Conceptualization. **P. Koumoutsakos:** Writing – review & editing, Writing – original draft, Supervision, Resources, Project administration, Methodology, Funding acquisition.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Data will be made available on request.

## Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT and Grammarly in order to enhance clarity and correct typographical errors. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

## Acknowledgments

PRV and PK are thankful to the Swiss National Supercomputing Center (CSCS) providing the necessary computational resources under project IDs s929 and s930.

## Appendix A. Lemma 1: Recursive relationship and summation formula for sequences

In this section, we will provide a proof of Lemma 1 by induction. We have that:

$$a_t = b_t + c_t \cdot a_{t-1} \tag{21}$$

and we want to show that:

$$a_t = b_t + \sum_{i=1}^{t-1}\Big(\prod_{j=i+1}^{t} c_j\Big) b_i. \tag{22}$$

First we start with the base case. For $t = 1$, we have from Eq. (21) that

$$a_1 = b_1 + c_1 \cdot a_0. \tag{23}$$

Since $a_0 = 0$, this simplifies to:

$$a_1 = b_1. \tag{24}$$

This satisfies the base case as the right-hand side of Eq. (22) reduces to $b_1$ when $t = 1$.

Next, we follow the inductive step. Assume the lemma holds for some $t = k$, i.e.,

$$a_k = b_k + \sum_{i=1}^{k-1}\left(\prod_{j=i+1}^{k} c_j\right) b_i. \tag{25}$$

We proceed by showing it holds for $t = k + 1$:

$$a_{k+1} = b_{k+1} + c_{k+1} \cdot a_k. \tag{26}$$

Using the induction hypothesis, we substitute for $a_k$:

$$\begin{aligned}
a_{k+1} &= b_{k+1} + c_{k+1} \cdot \left(b_k + \sum_{i=1}^{k-1}\left(\prod_{j=i+1}^{k} c_j\right) b_i\right)\\
&= b_{k+1} + c_{k+1} \cdot b_k + c_{k+1} \cdot \sum_{i=1}^{k-1}\left(\prod_{j=i+1}^{k} c_j\right) b_i\\
&= b_{k+1} + c_{k+1} \cdot b_k + \sum_{i=1}^{k-1}\left(\prod_{j=i+1}^{k+1} c_j\right) b_i\\
&= b_{k+1} + \sum_{i=k}^{k} c_{i+1} \cdot b_i + \sum_{i=1}^{k-1}\left(\prod_{j=i+1}^{k+1} c_j\right) b_i\\
&= b_{k+1} + \underbrace{\sum_{i=k}^{k}\left(\prod_{j=i+1}^{k+1} c_j\right)}_{c_{k+1}} \cdot b_i + \sum_{i=1}^{k-1}\left(\prod_{j=i+1}^{k+1} c_j\right) b_i\\
&= b_{k+1} + \sum_{i=1}^{k}\left(\prod_{j=i+1}^{k+1} c_j\right) b_i.
\end{aligned}$$

This concludes the inductive step, confirming that the property is maintained for $t = k + 1$. By the principle of mathematical induction, the lemma is therefore proven true for all applicable $t$, as asserted.

## Appendix B. Gradients in linear RNN without backpropagation through $x_t$

In this section, we evaluate the expressions of $o_2$, $o_3$, and $\partial o_2/\partial w_o$, $\partial o_3/\partial w_o$ for a linear RNN without backpropagation through the input $x_t$ (backpropagation through $x_t$ is turned off), as per BPTT-SS [57]. When the gradient is disabled for $x_t$, the system does not adjust based on the influence of $o_{t-1}$ on $x_t$. This means that $x_t$ is treated like a number $d_t$. $d_t$ can be set to its value $o_{t-1}$ after gradient computation. For this reason, we solve the following system of recursive equations, and compute $o_t$ for $t \in \{1, 2, 3\}$:

$$\begin{aligned}
h_t &= w_x x_t + w_h h_{t-1},\\
o_t &= w_o h_t,
\end{aligned}$$

with initial conditions:

$$\begin{aligned}
&h_0 = 0,\\
&x_t = d_t, \text{ for all } t > 0.
\end{aligned}$$

We get:

$$\begin{aligned}
h_1 &= w_x d_1,\\
h_2 &= w_x d_2 + w_h w_x d_1,\\
h_3 &= w_x d_3 + w_h w_x d_2 + w_h^2 w_x d_1,\\
o_2 &= w_o h_2,\\
o_3 &= w_o h_3.
\end{aligned}$$

Substituting the expressions for $h_2$ and $h_3$ into the equations for $o_2$ and $o_3$, we obtain:

$$\begin{aligned}
o_2 &= w_o w_x d_2 + w_o w_h w_x d_1,\\
o_3 &= w_o w_x d_3 + w_o w_x w_h d_2 + w_o w_h^2 w_x d_1,
\end{aligned}$$

Evaluating the gradients with respect to $w_o$, we get:

$$\begin{aligned}
\frac{\partial o_2}{\partial w_o} &= w_x d_2 + w_h w_x d_1,\\
\frac{\partial o_3}{\partial w_o} &= w_x d_3 + w_h w_x d_2 + w_h^2 w_x d_1,
\end{aligned} \tag{27}$$

Now, we can use the expressions for $d_2$:

$$d_2 = w_o h_1 = w_o w_x d_1$$

and $d_3$:

$$d_3 = w_o h_2 = w_o w_x d_2 + w_o w_h w_x d_1 = w_o^2 w_x^2 d_1 + w_o w_h w_x d_1 \tag{28}$$

Finally replacing expressions for $d_2$ and $d_3$ in the gradients Eq. (27) we get:

$$\begin{aligned}\frac{\partial o_2}{\partial w_o} &= w_o w_x^2 d_1 + w_h w_x d_1,\\ \frac{\partial o_3}{\partial w_o} &= w_o^2 w_x^3 d_1 + 2 w_o w_h w_x^2 d_1 + w_h^2 w_x d_1,\end{aligned} \tag{29}$$

## Appendix C. Gradients in linear RNN with backpropagation through $x_t$

In this section, we evaluate the expressions of $o_2$, $o_3$, and $\partial o_2/\partial w_o$, $\partial o_3/\partial w_o$ for a linear RNN with backpropagation through the input $x_t$, as per BPTT-SA. We compute the expression of $o_t$ step-by-step for $t = 1$, $t = 2$, and $t = 3$. For $t = 1$, we get:

$$h_1 = w_x d_1, \quad o_1 = w_o h_1 = w_o w_x d_1. \tag{30}$$

For $t = 2$:

$$\begin{aligned} x_2 &= o_1 = w_o w_x d_1,\\ h_2 &= w_x x_2 + w_h h_1 = w_o w_x^2 d_1 + w_h w_x d_1,\\ o_2 &= w_o h_2 = w_o^2 w_x^2 d_1 + w_o w_h w_x d_1.\end{aligned}$$

Next, for $t = 3$ we get:

$$\begin{aligned} x_3 &= o_2 = w_o^2 w_x^2 d_1 + w_o w_h w_x d_1,\\ h_3 &= w_x x_3 + w_h h_2 = w_o^2 w_x^3 d_1 + 2 w_o w_h w_x^2 d_1 + w_h^2 w_x d_1\\ o_3 &= w_o h_3 = w_o^3 w_x^3 d_1 + 2 w_o^2 w_h w_x^2 d_1 + w_o w_h^2 w_x d_1\end{aligned}$$

So we get:

$$o_3 = w_o^3 w_x^3 d_1 + 2 w_o^2 w_h w_x^2 d_1 + w_o w_h^2 w_x d_1. \tag{31}$$

By evaluating the gradients with respect to $w_o$ we finally get:

$$\begin{aligned}\frac{\partial o_2}{\partial w_o} &= 2 w_o w_x^2 d_1 + w_h w_x d_1, .\\ \frac{\partial o_3}{\partial w_o} &= 3 w_o^2 w_x^3 d_1 + 4 w_o w_h w_x^2 d_1 + w_h^2 w_x d_1.\end{aligned} \tag{32}$$

The terms of the gradients when backpropagation through $x_t$ is enabled as per BPTT-SA expressed in Eq. (32), and the terms of the gradients of BPTT-SS expressed in Eq. (29) are similar. However, the factors of the terms are different in each case.

## Appendix D. Gradients in linear RNN with nonlinear output activation without backpropagation through $x_t$

In this section, we evaluate the expressions of the output $o_2$, and $\partial o_2/\partial w_o$, for a linear RNN with nonlinear output activation function without backpropagation through the input $x_t$ (backpropagation through $x_t$ is turned off), as per BPTT-SS [57]. For simplicity we assume that the nonlinear output activation function is tanh. When the gradient is disabled for $x_t$, the system does not adjust based on the influence of $o_{t-1}$ on $x_t$. This means that $x_t$ is treated like a number $d_t$. $d_t$ can be set to its value $o_{t-1}$ after gradient computation. For this reason, we solve the following system of recursive equations, and compute $o_t$ for $t \in \{1, 2, 3\}$:

$$\begin{aligned} h_t &= w_x x_t + w_h h_{t-1},\\ o_t &= \tanh(w_o h_t),\end{aligned}$$

with initial conditions:

$$\begin{aligned} h_0 &= 0,\\ x_t &= d_t, \text{ for all } t > 0.\end{aligned}$$

We get:

$$\begin{aligned} h_1 &= w_x d_1,\\ h_2 &= w_x d_2 + w_h w_x d_1,\end{aligned}$$

**Table 1**
Hyperparameter tuning in Darwin dataset.

| Hyperparameter | Values |
|---|---|
| Optimizer | Adam |
| Batch size | 32 |
| Initial learning rate | 0.0001 |
| Max Epochs | 2000 |
| Random Seed | $\{1, \ldots, 10\}$ |
| BPTT sequence length $L$ | 100 |
| Prediction horizon | 100 |
| Number of testing initial conditions | 32 |
| Number of LSTM layers | 1 |
| Size of LSTM layers | 100 |
| Activation of LSTM Cell | tanh |
| Scaling | $[0, 1]$ |

**Table 2**
Hyperparameter tuning in Mackey Glass system.

| Hyperparameter | Values |
|---|---|
| Optimizer | Adam |
| SNR | $\{10, 60\}$ |
| Batch size | 32 |
| Initial learning rate | 0.0005 |
| Max Epochs | 2000 |
| Random Seed | $\{1, \ldots, 10\}$ |
| BPTT sequence length $L$ | 40 |
| Prediction horizon | 896 |
| Number of testing initial conditions | 100 |
| Number of LSTM layers | 1 |
| Size of LSTM layers | 100 |
| Activation of LSTM Cell | tanh |
| Scaling | $[0, 1]$ |

$$\begin{aligned} h_3 &= w_x d_3 + w_h w_x d_2 + w_h^2 w_x d_1,\\ o_2 &= \tanh(w_o h_2),\\ o_3 &= \tanh(w_o h_3).\end{aligned}$$

Substituting the expressions for $h_2$ and $h_3$ into the equations for $o_2$ and $o_3$, we obtain:

$$o_2 = \tanh(w_o w_x d_2 + w_o w_h w_x d_1).$$

Evaluating the gradients with respect to $w_o$, we get:

$$\frac{\partial o_2}{\partial w_o} = \Big(1 - \tanh^2(w_o w_x d_2 + w_o w_h w_x d_1)\Big) \cdot (w_x d_2 + w_h w_x d_1). \tag{33}$$

Now, we can use the expressions for $d_2$:

$$d_2 = \tanh(w_o h_1) = \tanh(w_o w_x d_1),$$

and $d_3$:

$$\begin{aligned} d_3 &= \tanh(w_o h_2) = \tanh(w_o w_x d_2 + w_o w_h w_x d_1)\\ &= \tanh(w_o w_x \tanh(w_o w_x d_1) + w_o w_h w_x d_1).\end{aligned} \tag{34}$$

Finally replacing expressions for $d_2$ and $d_3$ in the gradients given in Eq. (33) we get:

$$\begin{aligned}\frac{\partial o_2}{\partial w_o} &= \left(1 - \tanh^2\left(w_o w_x \tanh(w_o w_x d_1) + w_o w_h w_x d_1\right)\right)\\ &\quad \cdot \left(w_x \tanh(w_o w_x d_1) + w_h w_x d_1\right)\end{aligned} \tag{35}$$

Using the symbolic computation library [82], we can simplify the expression to:

$$\frac{\partial o_2}{\partial w_o} = \frac{w_x \left(d_1 w_h + \tanh\left(d_1 w_o w_x\right)\right)}{\cosh^2\left(w_o w_x \left(d_1 w_h + \tanh\left(d_1 w_o w_x\right)\right)\right)} \tag{36}$$

## Appendix E. Gradients in linear RNN with nonlinear output activation with backpropagation through $x_t$

In this section, we evaluate the expressions of $o_2$, and $\partial o_2/\partial w_o$, for a linear RNN with tanh output activation with backpropagation

**Table 3**
Architecture of CNN of CNN-RNNs in Navier–Stokes dataset.

| Layer | Encoder | Decoder |
|---|---|---|
| 1 | ZeroPad2d(padding=(5, 5, 5, 5), value=0.0) | Upsample(scale_factor=2.0, mode=bilinear) |
| 2 | Conv2d(3, 5, kernel_size=(11, 11), stride=(1, 1)) | ConvTranspose2d(1, 2, kernel_size=(3, 3), stride=(1, 1), padding=[1, 1]) |
| 3 | AvgPool2d(kernel_size=2, stride=2, padding=0) | CELU(alpha=1.0) |
| 4 | CELU(alpha=1.0) | BatchNorm2d(2, eps=1e−05, momentum=0.1, affine=False) |
| 5 | BatchNorm2d(5, eps=1e−05, momentum=0.1, affine=False) | Upsample(scale_factor=2.0, mode=bilinear) |
| 6 | ZeroPad2d(padding=(4, 4, 4, 4), value=0.0) | ConvTranspose2d(2, 20, kernel_size=(3, 3), stride=(1, 1), padding=[1, 1]) |
| 7 | Conv2d(5, 10, kernel_size=(9, 9), stride=(1, 1)) | CELU(alpha=1.0) |
| 8 | AvgPool2d(kernel_size=2, stride=2, padding=0) | BatchNorm2d(20, eps=1e−05, momentum=0.1, affine=False) |
| 9 | CELU(alpha=1.0) | Upsample(scale_factor=2.0, mode=bilinear) |
| 10 | BatchNorm2d(5, eps=1e−05, momentum=0.1, affine=False) | ConvTranspose2d(20, 10, kernel_size=(7, 7), stride=(1, 1), padding=[3, 3]) |
| 11 | ZeroPad2d(padding=(3, 3, 3, 3), value=0.0) | CELU(alpha=1.0) |
| 12 | Conv2d(10, 20, kernel_size=(7, 7), stride=(1, 1)) | BatchNorm2d(10, eps=1e−05, momentum=0.1, affine=False) |
| 13 | AvgPool2d(kernel_size=2, stride=2, padding=0) | Upsample(scale_factor=2.0, mode=bilinear) |
| 14 | Conv2d(5, 10, kernel_size=(9, 9), stride=(1, 1)) | ConvTranspose2d(10, 5, kernel_size=(9, 9), stride=(1, 1), padding=[4, 4]) |
| 15 | BatchNorm2d(20, eps=1e−05, momentum=0.1, affine=False) | CELU(alpha=1.0) |
| 16 | ZeroPad2d(padding=(1, 1, 1, 1), value=0.0) | BatchNorm2d(5, eps=1e−05, momentum=0.1, affine=False) |
| 17 | Conv2d(20, 2, kernel_size=(3, 3), stride=(1, 1)) | Upsample(scale_factor=2.0, mode=bilinear) |
| 18 | AvgPool2d(kernel_size=2, stride=2, padding=0) | ConvTranspose2d(5, 3, kernel_size=(11, 11), stride=(1, 1), padding=[5, 5]) |
| 19 | CELU(alpha=1.0) | 0.5 + 0.5 Tanh() |
| 20 | BatchNorm2d(20, eps=1e−05, momentum=0.1, affine=False) | |
| 21 | ZeroPad2d(padding=(1, 1, 1, 1), value=0.0) | |
| 22 | Conv2d(20, 2, kernel_size=(3, 3), stride=(1, 1)) | |
| 23 | AvgPool2d(kernel_size=2, stride=2, padding=0) | |
| 24 | CELU(alpha=1.0) | |
| Latent | $z \in \mathbb{R}^5$ | |
| Scaling | $[0, 1]$ | |

through the input $x_t$, as per BPTT-SA. We compute the expression of $o_t$ step-by-step for $t = 1$, $t = 2$, and $t = 3$. For $t = 1$, we get:

$$h_1 = w_x d_1,$$
$$o_1 = \tanh(w_o h_1) = \tanh(w_o w_x d_1).$$

For $t = 2$:

$$x_2 = o_1 = \tanh(w_o w_x d_1),$$
$$h_2 = w_x x_2 + w_h h_1 = w_x \tanh(w_o w_x d_1) + w_h w_x d_1$$
$$o_2 = \tanh(w_o h_2) = \tanh\Big(w_o w_x \tanh(w_o w_x d_1) + w_o w_h w_x d_1\Big)$$

In the following, we are going to evaluate the gradient of $o_2$ with respect to $w_o$. For this, we are using the auxiliary variable $u$:

$$\begin{aligned} u &= w_o w_x \tanh(w_o w_x d_1) + w_o w_h w_x d_1 \\ o_2 &= \tanh(u) \end{aligned} \tag{37}$$

By applying the chain rule, we get:

$$\frac{\partial o_2}{\partial w_o} = \frac{\partial \tanh(u)}{\partial u} \cdot \frac{\partial u}{\partial w_o} \tag{38}$$

Where we have that:

$$\frac{\partial \tanh(u)}{\partial u} = 1 - \tanh^2(u) \tag{39}$$

and the partial derivative of $u$ with respect to $w_o$ is:

$$\frac{\partial u}{\partial w_o} = w_x \tanh(w_o w_x d_1) + w_o w_x \cdot (1 - \tanh^2(w_o w_x d_1)) \cdot w_x d_1 + w_h w_x d_1. \tag{40}$$

In total we get:

$$\begin{aligned} \frac{\partial o_2}{\partial w_o} &= \frac{\partial \tanh(u)}{\partial u} \cdot \frac{\partial u}{\partial w_o} \implies \\ \frac{\partial o_2}{\partial w_o} &= \left(1 - \tanh^2\Big(w_o w_x \tanh(w_o w_x d_1) + w_o w_h w_x d_1\Big)\right) \cdot \\ &\quad \cdot \left(w_x \tanh(w_o w_x d_1) + w_o w_x \cdot (1 - \tanh^2(w_o w_x d_1)) \cdot w_x d_1 + w_h w_x d_1\right) \end{aligned} \tag{41}$$

Using the symbolic computation library [82], we finally arrive to:

$$\frac{\partial o_2}{\partial w_o} = \frac{w_x \left(d_1 w_h + \frac{d_1 w_o w_x}{\cosh^2(d_1 w_o w_x)} + \tanh(d_1 w_o w_x)\right)}{\cosh^2\Big(w_o w_x \big(d_1 w_h + \tanh(d_1 w_o w_x)\big)\Big)} \tag{42}$$

By comparing the gradients when backpropagation through $x_t$ is enabled as per BPTT-SA expressed in Eq. (42), and the gradients of BPTT-SS expressed in Eq. (36), we notice that an additional term appears on the nominator.

## Appendix F. Hyperparameters for Darwin sea level temperatures modeling

The hyperparameters for the networks employed in the Darwin dataset are given in Table 1.

## Appendix G. Hyperparameters for Mackey Glass modeling

The hyperparameters for the networks employed in the Mackey Glass system are given in Table 2.

## Appendix H. Hyperparameters for Navier Stokes flow modeling

The hyperparameters for the networks employed in the Navier–Stokes dataset are given in Table 5 for ConvRNNs and in Table 4 for CNN-RNNs. The autoencoder of CNN-RNNs is composed of consecutive Convolutional layers, Average pooling, CELU activation, and Batch-Norm layers. The exact architecture is given in Table 3. The autoencoder is reducing the dimensionality on a $z \in \mathbb{R}^5$ latent space. An LSTM with 24 units is predicting on this latent space.

**Table 4**
Hyperparameters of CNN-RNNs in Navier–Stokes dataset.

| Hyperparameter | Values |
|---|---|
| Optimizer | Adam |
| Batch size | 32 |
| Initial learning rate | 0.001 |
| Max Epochs | 1000 |
| Random Seed | $\{1,2,3,4\}$ |
| BPTT sequence length $L$ | 20 |
| Prediction horizon | 1000 |
| Number of testing initial conditions | 10 |
| RNN Cell | LSTM |
| Number of RNN layers | 1 |
| Size of RNN layers | 24 |
| Scaling | $[0,1]$ |

**Table 5**
Hyperparameters of ConvRNNs in Navier–Stokes dataset.

| Hyperparameter | Values |
|---|---|
| Optimizer | Adam |
| Batch size | 16 |
| Initial learning rate | 0.0001 |
| Max Epochs | 5000 |
| Random Seed | $\{1,2,3\}$ |
| BPTT sequence length $L$ | 50 |
| Prediction horizon | 1000 |
| Number of testing initial conditions | 10 |
| RNN Cell | LSTM |
| Number of RNN layers | 1 |
| Size of RNN layers | 8 |
| Kernel size | 5 |
| Scaling | $[0,1]$ |

## References

[1] P. Baldi, P. Sadowski, D. Whiteson, Searching for exotic particles in high-energy physics with deep learning, Nat. Commun. 5 (1) (2014) 1–9.
[2] B. Lusch, J.N. Kutz, S.L. Brunton, Deep learning for universal linear embeddings of nonlinear dynamics, Nat. Commun. 9 (1) (2018) 1–10.
[3] G. Novati, L. Mahadevan, P. Koumoutsakos, Controlled gliding and perching through deep-reinforcement-learning, Phys. Rev. Fluids 4 (9) (2019) 093902.
[4] S.L. Brunton, B.R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annu. Rev. Fluid Mech. 52 (2020) 477–508.
[5] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, et al., Exascale deep learning for climate analytics, in: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2018, pp. 649–660.
[6] H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, T. Blaschke, The rise of deep learning in drug discovery, Drug Discov. Today 23 (6) (2018) 1241–1250.
[7] B. Alipanahi, A. Delong, M.T. Weirauch, B.J. Frey, Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning, Nature Biotechnol. 33 (8) (2015) 831–838.
[8] P.J. Werbos, Generalization of backpropagation with application to a recurrent gas market model, Neural Netw. 1 (4) (1988) 339–356.
[9] J.L. Elman, Finding structure in time, Cogn. Sci. 14 (2) (1990) 179–211.
[10] P.J. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE 78 (10) (1990) 1550–1560.
[11] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, S. Khudanpur, Extensions of recurrent neural network language model, in: 2011 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, IEEE, 2011, pp. 5528–5531.
[12] W. Yin, K. Kann, M. Yu, H. Schütze, Comparative study of CNN and RNN for natural language processing, 2017, arXiv preprint arXiv:1702.01923.
[13] J. Koutnik, K. Greff, F. Gomez, J. Schmidhuber, A clockwork rnn, in: International Conference on Machine Learning, PMLR, 2014, pp. 1863–1871.
[14] K. Gregor, I. Danihelka, A. Graves, D.J. Rezende, D. Wierstra, Draw: A recurrent neural network for image generation, 2015, arXiv preprint arXiv:1502.04623.
[15] A.v.d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, Wavenet: A generative model for raw audio, 2016, arXiv preprint arXiv:1609.03499.
[16] A.M. Ahmad, S. Ismail, D. Samaon, Recurrent neural network with backpropagation through time for speech recognition, in: IEEE International Symposium on Communications and Information Technology, 2004, Vol. 1, ISCIT 2004, IEEE, 2004, pp. 98–102.
[17] W. Lim, D. Jang, T. Lee, Speech emotion recognition using convolutional and recurrent neural networks, in: 2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, APSIPA, IEEE, 2016, pp. 1–4.
[18] J. Anderson, I. Kevrekidis, R. Rico-Martinez, A comparison of recurrent training algorithms for time series analysis and system identification, Comput. Chem. Eng. 20 (1996) S751–S756.
[19] A.K. Rout, P.K. Dash, R. Dash, R. Bisoi, Forecasting financial time series using a low complexity recurrent neural network and evolutionary learning approach, J. King Saud Univ.-Comput. Inf. Sci. 29 (4) (2017) 536–552.
[20] P.R. Vlachas, W. Byeon, Z.Y. Wan, T.P. Sapsis, P. Koumoutsakos, Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks, Proc. R. Soc. A 474 (2213) (2018) 20170844.
[21] Z.Y. Wan, T.P. Sapsis, Machine learning the kinematics of spherical particles in fluid flows, J. Fluid Mech. 857 (2018) R2.
[22] N. Mohajerin, S.L. Waslander, Multistep prediction of dynamic systems with recurrent neural networks, IEEE Trans. Neural Netw. Learn. Syst. 30 (11) (2019) 3370–3383.
[23] B. Chang, M. Chen, E. Haber, E.H. Chi, AntisymmetricRNN: A dynamical system view on recurrent neural networks, 2019, arXiv preprint arXiv:1902.09689.
[24] O. San, R. Maulik, M. Ahmed, An artificial neural network framework for reduced order modeling of transient flows, Commun. Nonlinear Sci. Numer. Simul. 77 (2019) 271–287.
[25] R. Maulik, A. Mohan, B. Lusch, S. Madireddy, P. Balaprakash, D. Livescu, Time-series learning of latent-space dynamics for reduced-order model closure, Physica D 405 (2020) 132368.
[26] P.R. Vlachas, J. Pathak, B.R. Hunt, T.P. Sapsis, M. Girvan, E. Ott, P. Koumoutsakos, Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics, Neural Netw. (2020).
[27] N. Geneva, N. Zabaras, Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks, J. Comput. Phys. 403 (2020) 109056.
[28] F.P. Kemeth, T. Bertalan, N. Evangelou, T. Cui, S. Malani, I.G. Kevrekidis, Initializing LSTM internal states via manifold learning, Chaos 31 (9) (2021).
[29] R. Maulik, B. Lusch, P. Balaprakash, Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders, Phys. Fluids 33 (3) (2021).
[30] G. Uribarri, G.B. Mindlin, Dynamical time series embeddings in recurrent neural networks, Chaos Solitons Fractals 154 (2022) 111612.
[31] F.A. Gers, N.N. Schraudolph, J. Schmidhuber, Learning precise timing with LSTM recurrent networks, J. Mach. Learn. Res. 3 (Aug) (2002) 115–143.
[32] T.P. Lillicrap, A. Santoro, Backpropagation through time and the brain, Curr. Opin. Neurobiol. 55 (2019) 82–89.
[33] J. Walker, A. Gupta, M. Hebert, Patch to the future: Unsupervised visual prediction, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 3302–3309.
[34] M. Mathieu, C. Couprie, Y. LeCun, Deep multi-scale video prediction beyond mean square error, 2015, arXiv preprint arXiv:1511.05440.
[35] K. Fragkiadaki, P. Agrawal, S. Levine, J. Malik, Learning visual predictive models of physics for playing billiards, 2015, arXiv preprint arXiv:1511.07404.
[36] N. Srivastava, E. Mansimov, R. Salakhudinov, Unsupervised learning of video representations using lstms, in: International Conference on Machine Learning, 2015, pp. 843–852.
[37] J. Oh, X. Guo, H. Lee, R.L. Lewis, S. Singh, Action-conditional video prediction using deep networks in atari games, Adv. Neural Inf. Process. Syst. 28 (2015) 2863–2871.
[38] L. Castrejon, N. Ballas, A. Courville, Improved conditional vrnns for video prediction, in: Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 7608–7617.
[39] J. Chung, K. Kastner, L. Dinh, K. Goel, A.C. Courville, Y. Bengio, A recurrent latent variable model for sequential data, Adv. Neural Inf. Process. Syst. 28 (2015) 2980–2988.
[40] J. Pathak, Z. Lu, B.R. Hunt, M. Girvan, E. Ott, Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data, Chaos 27 (12) (2017) 121102.
[41] X. Shi, D.Y. Yeung, Machine learning for spatiotemporal sequence forecasting: A survey, 2018, arXiv preprint arXiv:1808.06865.
[42] Z.Y. Wan, P. Vlachas, P. Koumoutsakos, T. Sapsis, Data-assisted reduced-order modeling of extreme events in complex dynamical systems, PLoS One 13 (5) (2018) e0197704.
[43] J. Pathak, B. Hunt, M. Girvan, Z. Lu, E. Ott, Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach, Phys. Rev. Lett. 120 (2) (2018) 024102.
[44] X. Shi, Z. Chen, H. Wang, D.Y. Yeung, W.K. Wong, W.c. Woo, Convolutional LSTM network: A machine learning approach for precipitation nowcasting, Adv. Neural Inf. Process. Syst. 28 (2015) 802–810.
[45] S. Wiewel, M. Becher, N. Thuerey, Latent space physics: Towards learning the temporal evolution of fluid flow, 38, (2) Wiley Online Library, 2019, pp. 71–82,
[46] P.R. Vlachas, G. Arampatzis, C. Uhler, P. Koumoutsakos, Multiscale simulations of complex systems by learning their effective dynamics, Nat. Mach. Intell. 4 (4) (2022) 359–366.
[47] P.J. Blonigan, M. Farazmand, T.P. Sapsis, Are extreme dissipation events predictable in turbulent fluid flows? Phys. Rev. Fluids 4 (4) (2019) 044606.

[48] Y. Li, R. Yu, C. Shahabi, Y. Liu, Diffusion convolutional recurrent neural network: Data-driven traffic forecasting, 2017, arXiv preprint arXiv:1707.01926.

[49] A. Kumar, T. Islam, Y. Sekimoto, C. Mattmann, B. Wilson, Convcast: An embedded convolutional LSTM based architecture for precipitation nowcasting using satellite data, PLoS One 15 (3) (2020) e0230114.

[50] S. Rasp, P.D. Dueben, S. Scher, J.A. Weyn, S. Mouatadid, N. Thuerey, WeatherBench: A benchmark dataset for data-driven weather forecasting, 2020, arXiv preprint arXiv:2002.00469.

[51] X. Shi, Z. Gao, L. Lausen, H. Wang, D.Y. Yeung, W.-k. Wong, W.c. Woo, Deep learning for precipitation nowcasting: A benchmark and a new model, in: Advances in Neural Information Processing Systems, 2017, pp. 5617–5627.

[52] I. Sutskever, Training Recurrent Neural Networks, University of Toronto, Toronto, Canada, 2013.

[53] P. Le, W. Zuidema, Quantifying the vanishing gradient and long distance dependency problem in recursive neural networks and recursive LSTMs, 2016, arXiv preprint arXiv:1603.00423.

[54] S. Hochreiter, The vanishing gradient problem during learning recurrent neural nets and problem solutions, Internat. J. Uncertain. Fuzziness Knowledge-Based Systems 6 (02) (1998) 107–116.

[55] J. Mikhaeil, Z. Monfared, D. Durstewitz, On the difficulty of learning chaotic dynamics with RNNs, Adv. Neural Inf. Process. Syst. 35 (2022) 11297–11312.

[56] F. Schmidt, Generalization in generation: A closer look at exposure bias, 2019, arXiv preprint arXiv:1910.00292.

[57] S. Bengio, O. Vinyals, N. Jaitly, N. Shazeer, Scheduled sampling for sequence prediction with recurrent neural networks, Adv. Neural Inf. Process. Syst. 28 (2015) 1171–1179.

[58] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 41–48.

[59] M.A. Bucci, O. Semeraro, A. Allauzen, S. Chibbaro, L. Mathelin, Curriculum learning for data-driven modeling of dynamical systems, Eur. Phys. J. E 46 (3) (2023) 12.

[60] J. Miller, M. Hardt, When recurrent models don't need to be recurrent, 4 (2018) arXiv preprint arXiv:1805.10369.

[61] A.v.d. Oord, N. Kalchbrenner, K. Kavukcuoglu, Pixel recurrent neural networks, 2016, arXiv preprint arXiv:1601.06759.

[62] K. Gregor, I. Danihelka, A. Mnih, C. Blundell, D. Wierstra, Deep autoregressive networks, in: International Conference on Machine Learning, PMLR, 2014, pp. 1242–1250.

[63] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.

[64] J.L. Lin, C.W. Granger, Forecasting from non-linear models in practice, J. Forecast. 13 (1) (1994) 1–9.

[65] P. Teutsch, P. Mäder, Flipped classroom: Effective teaching for time series forecasting, 2022, arXiv preprint arXiv:2210.08959.

[66] M. Brenner, F. Hess, J.M. Mikhaeil, L.F. Bereska, Z. Monfared, P.C. Kuo, D. Durstewitz, Tractable dendritic RNNs for reconstructing nonlinear dynamical systems, in: International Conference on Machine Learning, PMLR, 2022, pp. 2292–2320.

[67] M. Brenner, G. Koppe, D. Durstewitz, Multimodal teacher forcing for reconstructing nonlinear dynamical systems, 2022, arXiv preprint arXiv:2212.07892.

[68] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014, arXiv preprint arXiv:1412.3555.

[69] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.

[70] A. Zhang, Z.C. Lipton, M. Li, A.J. Smola, Dive into deep learning, 2021, arXiv preprint arXiv:2106.11342.

[71] P.R. Vlachas, G. Arampatzis, C. Uhler, P. Koumoutsakos, Learning the effective dynamics of complex multiscale systems, 2020, arXiv preprint arXiv:2006.13431.

[72] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, 2019, pp. 8026–8037.

[73] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[74] A. Voelker, I. Kajić, C. Eliasmith, Legendre memory units: Continuous-time representation in recurrent neural networks, in: Advances in Neural Information Processing Systems, 2019, pp. 15570–15579.

[75] F.A. Gers, D. Eck, J. Schmidhuber, Applying LSTM to time series predictable through time-window approaches, in: Neural Nets WIRN Vietri-01, Springer, 2002, pp. 193–200.

[76] D. Harrison, N. Larkin, Darwin sea level pressure, 1876–1996: evidence for climate change? Geophys. Res. Lett. 24 (14) (1997) 1779–1782.

[77] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M.E. Rognes, G.N. Wells, The FEniCS project version 1.5, Arch. Numer. Softw. 3 (100) (2015).

[78] H. Petter Langtangen, A. Logg, Solving PDEs in Python: The FEniCS Tutorial I, Springer Nature, 2017.

[79] Z. Wang, A.C. Bovik, H.R. Sheikh, E.P. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Trans. Image Process. 13 (4) (2004) 600–612.

[80] S.S. Du, S.M. Kakade, R. Wang, L.F. Yang, Is a good representation sufficient for sample efficient reinforcement learning? 2019, arXiv preprint arXiv:1910.03016.

[81] J. Su, W. Byeon, F. Huang, J. Kautz, A. Anandkumar, Convolutional tensor-train LSTM for spatio-temporal learning, 2020, arXiv preprint arXiv:2002.09131.

[82] A. Meurer, C.P. Smith, M. Paprocki, O. Čertík, S.B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J.K. Moore, S. Singh, et al., Sympy: symbolic computing in python, PeerJ Comput. Sci. 3 (2017) e103.