Real-Time and Adaptive Reservoir Computing with an Application to Profile Prediction in Fusion Plasma

Azarakhsh Jalalvand*†, Joseph Abbate§¶, Rory Conlin†¶, Geert Verdoolaege‡, Egemen Kolemen†¶
*Dept. of Electronics and Information Systems, Ghent University, Ghent, B-9052, Belgium
†Dept. of Mechanical and Aerospace Engineering, Princeton University, Princeton, NJ 08544, USA
‡Dept. of Applied Physics, Ghent University, Ghent, B-9000, Belgium
§Dept. of Astrophysical Sciences, Princeton University, Princeton, NJ 08544, USA
¶Princeton Plasma Physics Laboratory, Princeton, NJ 08543, USA

***Abstract*—Nuclear fusion is a promising alternative to address the problem of sustainable energy production. The tokamak is an approach to fusion based on magnetic plasma confinement, constituting a complex physical system with many control challenges. We study the characteristics and optimization of Reservoir Computing Networks (RCNs) for real-time and adaptive prediction of plasma profiles in the DIII-D tokamak. Our experiments demonstrate that RCNs achieve comparable results to state-of-the-art (deep) convolutional neural networks and long short-term memory (LSTM) models, with a significantly easier and faster training procedure. This superiority allows for fast and frequent adaptation of the model to new situations, such as changing plasma conditions or different fusion devices.**

***Index Terms*—adaptive learning, condition monitoring, reservoir computing, nuclear fusion, tokamak plasma**

I. Introduction

The research on controlled nuclear fusion aims at the development of a source of power that is clean, safe and as good as inexhaustible. Among the various approaches, the tokamak configuration, based on magnetic confinement of a hot hydrogen isotope plasma, is currently the most advanced one. Understanding and control of the physical mechanisms governing the transport of heat and particles through the plasma are essential to optimize the plasma confinement, hence fusion performance, while ensuring machine operation within the design limits. In turn, this requires careful monitoring and control of the plasma properties, such as density and temperature throughout the plasma volume. In particular, model-based prediction of plasma properties, based on measurements of the current and past state of the plasma, has the potential to greatly facilitate maintaining the plasma in the desired state, using a variety of actuators.

Physics models are currently not always sufficiently accurate or computationally too demanding to be used for plasma control, particularly when certain plasma instabilities arise which may eventually lead to a complete loss of plasma confinement in a disruption [1], [2]. In this regard, data-driven techniques are being studied to simulate the evolution of important plasma properties and predict the plasma state on a very short time scale. In particular, artificial neural networks (ANN) have been extensively studied for plasma evolution monitoring tasks. Researchers at TEXT [3] DIII-D [4] ASDEX [5] and JET [6], [7] have presented proof-of-concept ANN-based models to monitor the plasma state and detect disruptions by using diagnostic data available in real time as input signals. Recently, a disruption predictor combining recurrent and convolutional neural networks has been developed by Kates-Harbeck, et. al [8]. The neural network has also been applied to analyze the tearing mode (TM) on JET [9]. Other machine learning methods, such as support vector machines [10], [11], discriminant analysis [12] and ensemble methods [13]–[15] have been studied as well with a view to disruption prediction.

The advanced machine learning models that have been used so far for plasma state prediction are usually developed by training, optimizing, and testing the model with large data sets. This is an expensive process for complex models, such as deep neural networks [8], which requires highly skilled individuals, large data sets, huge computational resources and capabilities, and a lot of time. Unfortunately, current neural network models are not adaptable to incremental changes in the environmental conditions, despite the intense investment of time and resources. For instance, it has proven difficult to generalize the performance of disruption predictors between operational conditions or from one fusion device to another. Without adaptability, the retraining of models must be performed by submitting new training data sets, again requiring a huge investment of resources and time.

To address the challenges of neural network training for plasma state prediction, in this study we employ adaptive reservoir computing networks (RCNs) [16]–[19]. To our knowledge, this is the first application of reservoir computing in the context of fusion research. RCNs have shown to be very effective for a variety of tasks, e.g., in the analysis of high dimensional data and time-evolving chaotic systems [20]–[23]. We demonstrate the performance of an RCN-based model for space-resolved prediction of density and temperature in the DIII-D tokamak, given information on the present plasma state and future information on the actuator settings. We describe in detail how to develop and tune the network to achieve the optimal performance. Furthermore, we investigate the potential of a model based on reservoir computing in real-

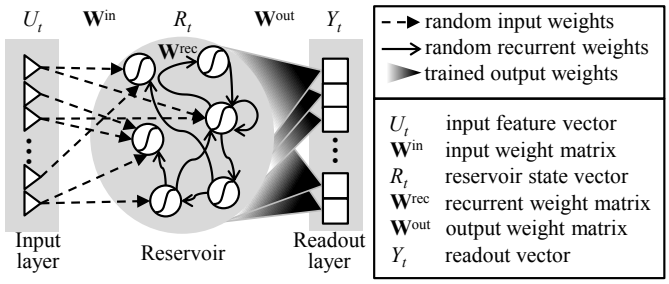

Fig. 1. A basic RCN consists of a reservoir and a readout layer. The reservoir is composed of interconnected nonlinear neurons with fixed random weights. The readout layer consists of linear neurons with trained weights.

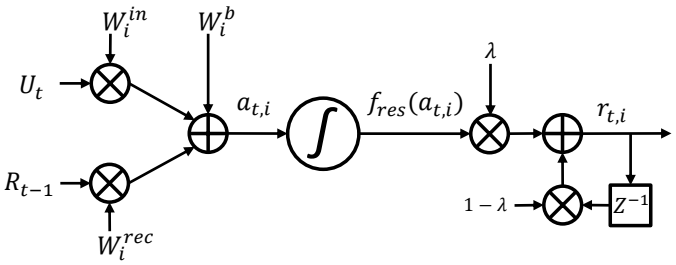

Fig. 2. Visualization of Equation (1) describing leaky integrator neuron i.

time adaptation to recent input. This is an important asset of this type of neural networks that discriminates them from alternative complex data-driven models, such as deep neural networks.

The rest of this paper is organized as follows. Section II gives an overview of reservoir computing networks. In Section III, we explore the main hyper-parameters of the RCN and study their impact on the reservoir states. Section IV describes the collected data set followed by the experimental results in SectionV. The paper ends with a brief conclusion and ideas for future work.

II. Reservoir Computing Network (RCN)

RCN is a neural network with two particular computational layers: (1) a hidden layer of recurrently interconnected nonlinear neurons, driven by inputs and by delayed feed-back of its activation and (2) an output layer of linear neurons, driven by the hidden neuron activation (Fig. 1). A fundamental point is that the input weights and the recurrent connection weights are initialized randomly, and only the output weights are optimized (trained) for solving the targeted problem.

The recurrently interconnected hidden neurons constitute a *reservoir* (a pool) of computational neurons. The reservoir can be viewed as a nonlinear dynamical system that analyzes a stream of inputs, e.g. time series data. The outputs are usually called *readouts* [24], so as to differentiate them unambiguously from the reservoir outputs. If U_t, R_t and Y_t represent the reservoir inputs, the reservoir outputs and the readouts at time t, the RCN equations can be written as follows:

$$R_t = (1-\lambda)R_{t-1} + \lambda f_{res}(\mathbf{W}^{in}U_t + \mathbf{W}^{rec}R_{t-1} + \mathbf{W}^b) \quad (1)$$

$$Y_t = \mathbf{W}^{out}R_t \quad (2)$$

with λ being a leaking rate between 0 and 1, with f_{res} being the nonlinear activation function of the reservoir neurons (we used *hyperbolic tangent* in this work) and with $\mathbf{W}^{in}$, $\mathbf{W}^{rec}$, $\mathbf{W}^b$ and $\mathbf{W}^{out}$ being the input, recurrent, bias and output weight matrices, respectively. Equation (1) and Fig.2 represent a leaky integration of the neuron activation.

The weights of the hidden neurons are fixed by means of a random process that is characterized by four parameters [25]: (1) α_U, the maximal absolute eigenvalue of the input weight matrix $\mathbf{W}^{in}$, (2) ρ, also known as *spectral radius*, the maximal absolute eigenvalue of the recurrent weight matrix $\mathbf{W}^{rec}$, (3) K^{in}, the number of inputs driving each reservoir neuron and (4) K^{rec}, the number of delayed reservoir outputs driving each reservoir neuron. The first two parameters control the relative importance of the inputs and the delayed reservoir outputs in the reservoir neuron activation. The latter two control the sparsity of the input and the recurrent weight matrices. $\mathbf{W}^b$ is also initialized randomly and re-scaled by α_b as a hyperparameter.

Any effective reservoir should at least have the so-called echo state property, stating that with time, the reservoir should forget the initial state it was in. That is also why a reservoir network was originally called an Echo State Network [24]. It was shown in [24] that the echo state property holds if ρ, the spectral radius of the recurrent weight matrix, is smaller than 1.

A. How does RCN work?

In essence, the reservoir can be seen as a random fixed projector that nonlinearly projects the input sample, represented by an input feature vector of size N^{in}, to a (usually much) higher-dimensional feature space of size N^{res}. Assuming that the complexity (e.g., the nonlinearity) of the given task and data is an obstacle to easily (e.g., linearly) separate the data samples, the hypothesis is that the reservoir projects the inputs ($\mathbf{U}$) to a new feature space $\mathbf{R}$ and facilitates the linear separation of the data samples. In Fig. 3, we provide a simple example which can help to understand how this projection work. It shows a small data set consisting of ten 1-d samples randomly labeled as two classes and the task is to train a linear classifier based on the the available data. Obviously there is no linear model to perfectly separate these two classes and the best model scores 6 out of 10.

By initializing a reservoir computing model of size 2, each of the 1-d samples is projected to a new 2-d feature space. Consequently, we are able to train a linear model on this space which can better classify the the data and achieve a score of 8. Although a valid concern would be the sensitivity of the performance to the random initialization of reservoir weights, studies show that the performance of the model is quite stable for sufficiently large reservoirs [26].

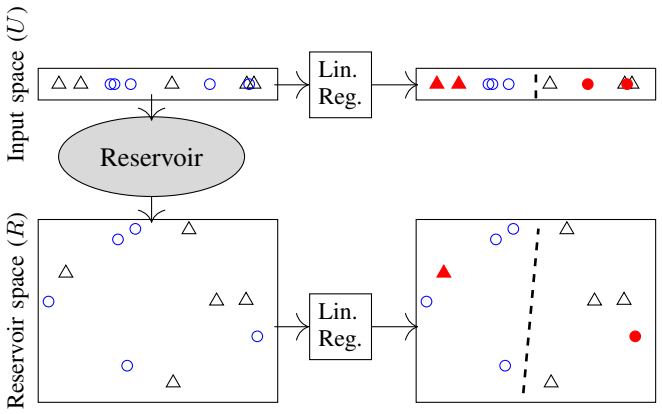

Fig. 3. An example of applying a 2-node reservoir to a 1-d data set. *top left* is the data feature space and the *top right* shows the outcome of linear regression to classify the samples. *bottom left* is the result of feeding the above samples to the reservoir and *bottom right* shows the optimal regression on this new space. The black dashed line shows the hyperplane determined by the regression model and the miss-classified samples are marked with red.

B. Training

The aim of the training is to find the output weights that minimize the mean squared difference between the readouts Y_t and their desired values D_t across N^{tr} available training examples. Introducing the matrices $\mathbf{R}$ and $\mathbf{D}$ with columns R_t and D_t respectively, the output weights are the solution of a regularized Tikhonov regression problem [27]:

$$\mathbf{W}^{out} = \underset{\hat{W}^{out}}{arg\,min} \left(\frac{1}{N^{tr}} \left|\left| \hat{\mathbf{W}}^{out}\mathbf{R} - \mathbf{D} \right|\right|^2 + \epsilon \left|\left| \hat{\mathbf{W}}^{out} \right|\right|^2 \right) \tag{3}$$

with ϵ being the regularization parameter which is intended to prevent overfitting to the training data. The solution is obtained in a closed-form [28] as

$$\mathbf{W}^{out} = (\mathbf{R}^\intercal \mathbf{R} + \epsilon\, \mathbf{I})^{-1} (\mathbf{R}^\intercal\, \mathbf{D}), \tag{4}$$

with $\mathbf{I}$ representing the identity matrix and $\mathbf{A}^{-1}$ the Moore-Penrose pseudo inverse of $\mathbf{A}$ [29]. Algorithm 1 presents the steps of training an RCN.

C. Adaptation

Most predictive models are developed under the assumption that training and testing data are generated from a stationary process. However, this assumption often does not hold true in practice. For instance changes in either the process configuration or machine calibration are the usual sources of concept drift in the data which directly influences the performance of the data-driven prediction approaches [30]. In fusion devices, such drifts can arise when transitioning to a new regime of plasma operation, or when testing a model on a different machine. As a result, there is a need to retrain or adapt the deployed models to the changes of the environment. While retraining is a very complex and expensive process in many state-of-the-art models such as deep neural networks, RCN's easy one-shot training process makes it a very appealing alternative.

Algorithm 1 Training RCN

1: **procedure** INITIALIZE $\mathbf{W}^{in}$, $\mathbf{W}^{rec}$, $\mathbf{W}^{b}$
2: **procedure** EXECUTE THE RCN ON THE SAMPLES
3: $\boldsymbol{\mathcal{R}} \leftarrow$ zero array of shape $(N^{res}+1) \times (N^{res}+1)$
4: $\boldsymbol{\mathcal{D}} \leftarrow$ zero array of shape $(N^{res}+1) \times N^{out}$
5: $\mathcal{N}^{tr} \leftarrow 0$
6: **for** `each training shot of length` T **do**
7: $U \leftarrow$ feature array of shape $N^{in} \times T$
8: $D \leftarrow$ target array of shape $N^{out} \times T$
9: $R \leftarrow$ zero array of shape $N^{res} \times T$
10: $R_0 \leftarrow$ zero array of shape $N^{res} \times 1$
11: **for** t: `1 to` T **do**
12: $R_t \leftarrow \tanh(\mathbf{W}^{in} \times U_t + \mathbf{W}^{rec} \times R_t + \mathbf{W}^{b})$
13: $R_t \leftarrow (1-\lambda)R_{t-1} + \lambda R_t$
14: add a row of 1s to R for bias
15: $\mathcal{N}^{tr} \leftarrow \mathcal{N}^{tr} + T$
16: $\boldsymbol{\mathcal{R}} \leftarrow \boldsymbol{\mathcal{R}} + R^\intercal \times R$
17: $\boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{D}} + R^\intercal \times D$
18: **procedure** TRAIN THE OUTPUT WEIGHTS
19: $\mathbf{W}^{out} = (\boldsymbol{\mathcal{R}} + \epsilon\, \mathbf{I})^{-1}(\boldsymbol{\mathcal{D}})$

Since the readout nodes are linear, the linear transformation of readouts is equivalent to a linear transformation of the readout node parameters (weights). Learning the latter transformation can be formulated as training the readouts with the original training data supplemented with the adaptation data. Based on Algorithm 1, instead of storing the whole training data we only need to store the matrices $\boldsymbol{\mathcal{R}}$ and $\boldsymbol{\mathcal{D}}$ along with the scalar $\mathcal{N}^{tr}$ after the training. By collecting $\boldsymbol{\mathcal{R}}_A$, $\boldsymbol{\mathcal{D}}_A$ and $\mathcal{N}_A^{tr}$ from the new (adaptation) data, the adapted readout weights can be obtained as

$$\mathbf{W}^{out} = ((1-\gamma)\boldsymbol{\mathcal{R}} + \gamma\boldsymbol{\mathcal{R}}_A + \epsilon\, \mathbf{I})^{-1}((1-\gamma)\boldsymbol{\mathcal{D}} + \gamma\boldsymbol{\mathcal{D}}_A) \tag{5}$$

where γ is a factor that controls how much the adaptation data contribute to these weights. In that regard $\gamma = 1$ is a special case in which the impact of the old samples are discarded when they become obsolete and therefore misleading. The cost of adaptation is discussed in Section V.

D. Parallel Training

Algorithm 1 suggests that the main steps of training RCN are (1) initializing the input, recurrent and bias weight matrices, (2) accumulating $\mathcal{R}$ and $\mathcal{D}$ for the samples in the training set, and (3) applying ridge regression on the final accumulated $\mathcal{R}$ and $\mathcal{D}$. Assuming that executing the RCN (lines 10-17 of the algorithm) for each training sample begins from the initial state of $R_0 = 0$, we can conclude that calculating $\mathcal{R}$ and $\mathcal{D}$ for each sample (shot in this data set) is independent of the others. Therefore it is possible to split the training data set in to batches of samples, execute each batch in a separate processor and accumulate the aforementioned matrices afterwards to calculate the output weights (see Algorithm 2).

Algorithm 2 Parallel training of RCN

1: Initialize $\mathbf{W}^{in}$, $\mathbf{W}^{rec}$, $\mathbf{W}^{b}$
2: Split the data set to G arbitrary groups and distribute them to the available processors
3: **for** `each training group` g **do**
4: Calculate $\boldsymbol{\mathcal{R}}_g$, $\boldsymbol{\mathcal{D}}_g$ and $\mathcal{N}_g^{tr}$ (See Algorithm 1 procedure 2)
5: **procedure** TRAIN THE OUTPUT WEIGHTS
6: $\mathcal{N}^{tr} = \sum \mathcal{N}_g^{tr}$
7: $\boldsymbol{\mathcal{R}} = \sum \boldsymbol{\mathcal{R}}_g$
8: $\boldsymbol{\mathcal{D}} = \sum \boldsymbol{\mathcal{D}}_g$
9: $\mathbf{W}^{out} = (\boldsymbol{\mathcal{R}} + \epsilon\, \mathbf{I})^{-1}(\boldsymbol{\mathcal{D}})$

III. HYPERPARAMETERS OF RCN

In this section, we analyze the impact of the important hyperparameters, namely the input scaling α_{U} and the spectral radius ρ along with the density of the $\mathbf{W}^{in}$ and $\mathbf{W}^{rec}$, the leakage λ and the bias scaling α_{b}. To that end, we feed a time-shifted unit impulse into a small RCN and observe the absolute values of the impulse responses inside the reservoir. Such a training-free analysis would provide a conceptual understanding of reservoir behavior independent of the use case and quality of the training data.

Density of the input and recurrent connections: Studies on different tasks show that the sparsity of the input and recurrent weights does not significantly influence the performance as long as the relative scales, α_U and ρ, are adjusted correctly [31], [32]. Nevertheless, it is of interest to verify whether the sparsity of the input and recurrent weights have observable impact on the reservoir activation. Figure 4 shows the impulse response of a 100-node reservoir with combinations of sparse and dense input and recurrent weights. These reservoirs have been supplied with an impulse input at time $t = 5$. While there is no clear impact on the shape and length of the activation (i.e., dynamical memory of reservoir), some arguments in favor of sparse connections are:

- In a fully connected network, all the nodes are triggered with all the input features and the difference in their activation only relies on the on the different connection weights given to each feature. On the other hand, in a sparsely connected network the activation of each node is different from the other not only because of the random weights, but also because each node is triggered by a different set of input features. This increases the diversity of information combination and reduces the adverse effects of missing data which could lead to more robustness of the model.
- Employing sparse connections has a significant impact on the hardware efficiency of the model, specially when dealing with multidimensional inputs and/or when very large reservoirs are required. For instance, $\mathbf{W}^{rec}$ of a 16000-node fully connected reservoir occupy around 2 GB of memory whereas such a matrix for the same reservoir with only 10 connections per node would occupy less than 2 MB.
- Moreover, if we define the sparsity of input connections by the number of input connections to each reservoir node (K^{in}) instead of a total random selection, the total number of non-zero elements in $\mathbf{W}^{in}$ will be $N^{res} \times K^{in}$. Therefore, the size of $\mathbf{W}^{in}$, and consequently the required computational resources, will be independent of the dimension of input features vector.

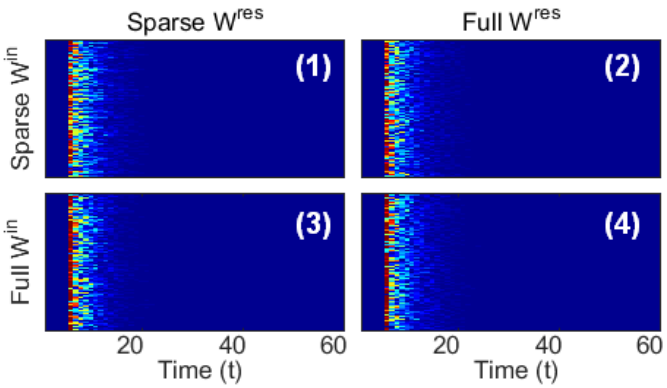

Fig. 4. Impulse response of a 100-node reservoir with (1) sparse input and recurrent connection matrices, (2) sparse input and full recurrent connection matrices, (3) full input and sparse recurrent connection matrices, (4) full input and recurrent connection matrices.

Memory concept of RCN: In a second experiment, we studied the impact of leaking rate (λ) and spectral radius (ρ) as the two parameters to control the memory of reservoirs. In theory, the leaky integration neurons (LINs) and the recurrent connections of the reservoir enable modeling of the short-term dynamics of motion. In order to give a visual idea of these dynamics, Fig. 5 shows the impulse response of a 100-node reservoir, with and without recurrent connections (controlled with ρ) and with and without LINs (controlled with λ). Obviously the system without LIN and recurrent connections has no means to remember the past, hence the response is also an impulse. Such a model is known as conventional Extreme Learning Machine (ELM) [33]. By replacing simple neurons with LINs, it is possible to provide a linear fading memory: the smaller the λ, the longer the effect of the past information but with weaker first response. In practice, this means that the state of each neuron will change faster or slower depending on the leakage λ. A reservoir with simple neurons and with recurrent connections benefits from a complex short-term dynamics. This information also fades out through time (if $\rho < 1$) but it is difficult to interpret, because of the nonlinearity of the neurons and random initialization of the the recurrent weights. The reservoir with LINs and recurrent connections benefits from both of these memory concepts and the combined information lasts longer than each individual ones.

Figure 5 also shows the impulse responses for $\rho = 1.05$. We can see that the reservoir states are decaying very slowly, and they are oscillating with a resonance frequency. For many tasks, it is indeed necessary to preserve the echo state property of reservoir and keep $\rho < 1$. However, in some cases the spectral radius can be larger than 1 [34].

Bias scaling: Another hyperparameter to be studied is the bias scaling α_b. The stable state of all reservoir neurons without bias is the center value of the nonlinear activation

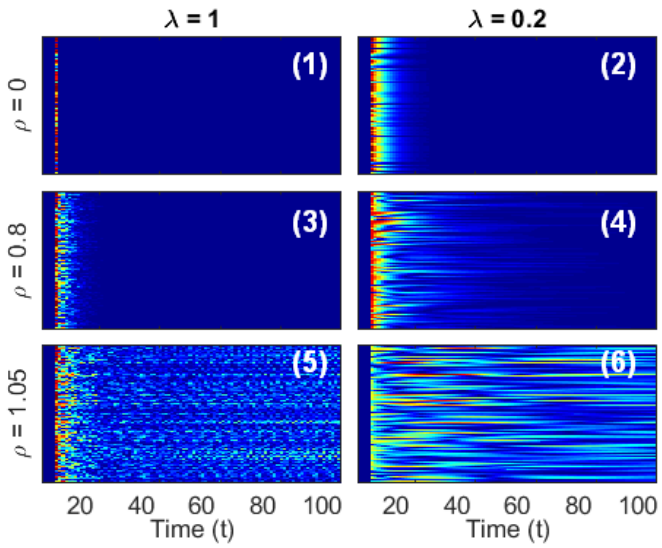

Fig. 5. Impulse response of a 100-node reservoir (1) with simple neurons and without recurrent connections, (2) with leaky integrated neurons and without recurrent connections, (3) with simple neurons and with recurrent connections, (4) with leaky integrated neurons and with recurrent connections. (5) and (6) show the unstable regime of the reservoir when $\rho > 1$

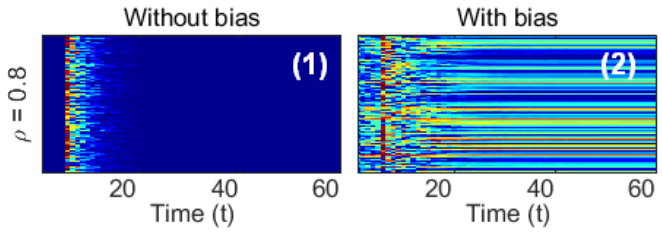

Fig. 6. Impulse response of a 100-node reservoir (1) with recurrent connections and without bias, (2) with recurrent connections and with bias. Bias helps to reach more nonlinear areas of the activation function by changing the stable state of the reservoir nodes.

function (tanh), i.e., 0. The bias scaling can help to reach more non-linear areas of the activation function by changing the stable state of the reservoir nodes. This is particularly useful in the tasks when the nonlinearity of the model plays an important role on the performance.

In Fig. 6, we present the impact of bias in a reservoir without leaky integration ($\lambda = 1$) but with a spectral radius of $\rho = 0.8$. The absolute value of the stable states of the reservoir neurons is approximately distributed in the activation range and each neuron has its own stable state. When new information from the input is passed to the reservoir neurons, this is the excitation point.

IV. Dataset

In this work, data was collected from the DIII-D tokamak, from experimental campaigns covering the years 2010 to 2018. The MDSplus data management software was used to load the data [35], alongside the OMFIT software framework for data preprocessing [36]. A new module has been developed for this task and is publicly available within OMFIT. A tokamak is an inherently pulsed device, with the device settings and plasma conditions determined by the experimental program. A plasma pulse (discharge or "shot") broadly consists of a ramping-up phase of the plasma current, a current flat-top phase and a current ramp-down. In this work, only data from the flat-top phase was used. DIII-D shots have a typical duration of the order of seconds.

All signals were resampled to a common 50 ms time base by averaging or nearest-neighbor interpolation. Our model considers data in 50 ms non-overlapping frames, a spacing large enough to smooth over most irrelevant signal variations, like modulations of the injected power. Each chunk corresponds to a single "timestep". Three types of data were considered:

- plasma profiles: cross-sections of plasma quantities like density and temperature. We consider radial profiles, from the center of the vacuum vessel up to the plasma boundary, at each time step over the shot leading to a 65-dimensional data vector per profile at time t.
- global plasma parameters (0-d signals) characterizing the overall plasma conditions, like (target) plasma current and density, total injected power, etc.
- actuator settings, i.e. plasma and machine parameters that allow the operator a certain amount of control over the plasma profiles.

In this work, the plasma state is determined jointly by the plasma profiles and global parameters. The full list of signals used in this work is shown in Table I.

	Signal name	Units
Profiles	Electron Density (n_e)	$10^{19}/\mathrm{m}^3$
	Electron Temperature (T_e)	keV
	Ion Rotation (Ω)	kHz
	Rotational Transform[1] (ι)	-
	Plasma Pressure (P)	Pa
Actuators	Injected Power	MW
	Injected Torque	N·m
	Target Current	A
	Target Density	$10^{19}/\mathrm{m}^3$
Scalar params	Top Triangularity	-
	Bottom Triangularity	-
	Plasma Elongation	-
	Plasma Volume	m^3
	Internal Inductance	μH
	Line Avg. Density	$10^{19}/\mathrm{m}^3$

TABLE I
SIGNALS COLLECTED FROM DIII-D TOKAMAK AND USED IN THE PROFILE PREDICTION NEURAL NETWORK.

Figure 7 provides an example of the time evolution of the various profiles in one plasma discharge at DIII-D.

The input for the model is the plasma state at the current time step, consisting of the measured profiles and global plasma parameters. In addition, a "proposal" is given for the value of the actuators at each of 4 time steps (200ms) into the future and the algorithm predicts the change in each of the profiles 200 ms (4 time steps) into the future. This 200 ms prediction window was chosen based on the typical energy confinement time (τ_E) at DIII-D, and was empirically found to be a period over which the profiles change noticeably while not so long that the future state cannot be reliably predicted. However, in Section V we also briefly study the performance

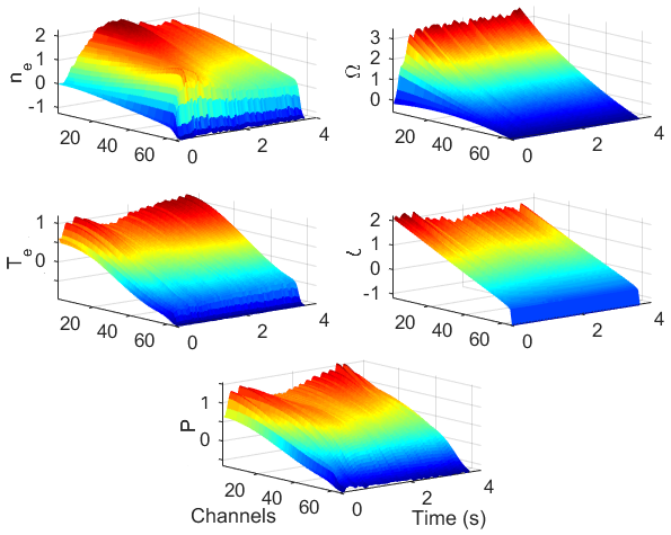

Fig. 7. Measured evolution of plasma profiles in DIII-D shot #154971.

of the RCN as a function of this prediction window.

The data from all shots were split into a 75% training set (2,750 shots longer than 2 s, corresponding to ca. 132,000 frames), 15% for validation (705 shots) and 10% for final testing (370 shots). Each signal was normalized by subtracting out the median value and dividing by the inter-quartile range.

V. Experimental Results

We report the performance of the prediction model using root mean squared errors (RMSE), based on the difference between the prediction of the model and the actual measurement for each signal, and normalized by the standard deviation of the actual data .

A. RCN Hyperparameter Tuning

In Section III we presented a conceptual understanding of the reservoir's hyperparameters consisting of $(K^{in}, K^{rec}, \alpha_U, \rho, \lambda)$ along with bias and the reservoir size. Setting up a suitable RCN has been studied in detail in [19] for the task of speech recognition. The empirical findings of that work shows that there are simple and comprehensible rules which allow to design a reservoir in a structured manner rather than a naive and time consuming grid search over all the parameters:

1) In order to optimize the hyperparameters, one can begin with a rather small size reservoir.
2) The input and recurrent weight matrices ($\mathbf{W}^{in}$ and $\mathbf{W}^{rec}$) can be very sparse. In particular, 5 to 10 elements per node are enough, regardless of the size of the reservoir and the input feature vector.
3) ρ and α_U together control the relative importance of the inputs and recurrent neuron activation. Therefore they can be tuned based on prior knowledge on the relation of these two activation (e.g., current input needs more weight than the past information) or a plain grid search.
4) The leaking rate λ can be tuned based on the minimum time (in scan steps) the reservoir output is expected to remain constant, i.e., the size of memory needed for the given task.
5) The last parameter to optimize is the size of the reservoir. The maximum size of the reservoir depends on the quality and quantity of the training data, hardware limitations, and the ideal processing time.

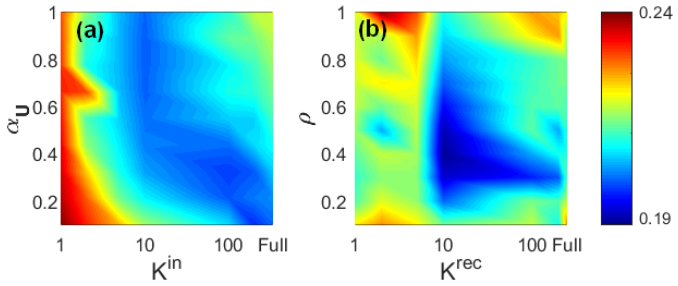

Fig. 8. RMSE on the validation set (a) as a function of α_U and K^{in} and (b) as a function of ρ and K^{rec}.

In order to optimize the input layer, we begin with a rather small reservoir of 250 nodes supplied with profiles, parameters and actuators as inputs to predict the profiles. As a result the input at each timestep is a vector of size $N^{in} = 335$ (5 profiles of 65 each + 6 parameters + 4 actuators) to predict all five profiles ($N^{out} = 325$).

Figure 8(a) shows the performance of an RCN without recurrent connections as a function of α_U and K^{in}. The results confirm that the density of input connections is indeed not a bottleneck as long as the input weight scale is chosen correctly. Also the linear relation of these parameters suggests that by changing one of them, the other one can be adjusted without sweeping. Figure 8(b) depicts the outcome of a similar experiment on the recurrent weights. According to these results, (1) adding recurrent connections improves the performance of the RCN and (2), a very sparse recurrent connection of only 10 inputs per neuron is enough to achieve the optimal performance.

In the next experiment we study the impact of each of the three input categories (profiles, actuators and scalar parameters) on the performance of the RCN model. According to Fig. 9, while actuators and scalars fail to individually predict the profiles, actuators play a more important role in contributing to the current profiles to predict the future of the profiles. Nevertheless, since the dimension of the input vector has negligible impact on the processing complexity of RCN, we feed all the available inputs to RCN.

Figure 10 presents the performance of an RCN as a function of λ and it shows that a rather small leakage ($\lambda = 0.85$) helps the reservoir to benefit from a short-term memory in predicting the profiles.

The last hyperparameter to be optimized is the size of the reservoir. According to Fig. 11 a reservoir of 1000 to 3000 nodes is the optimal configuration. The total number of trainable parameters is in fact the size of $\mathbf{W}^{out}$ or $(N^{res}+1)\times N^{out}$. Given that the proposed model is supposed to predict five profiles of dimension 65 for each, the 1000-node RCN has $325,325$ parameters which means that the overfitting

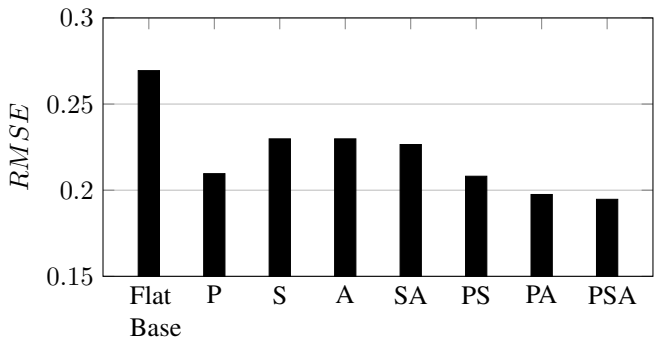

Fig. 9. Performance of the RCN as a function of the different combinations of inputs. P, S, and A refer to profiles, scalar parameters and actuators, respectively. Flat Baseline assumes that the signal at the intended future will be exactly the same as now.

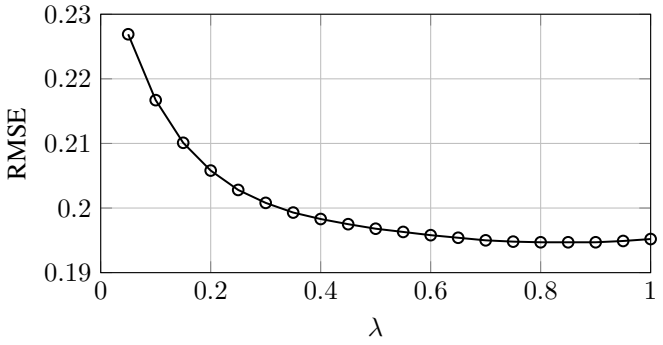

Fig. 10. Sweeping λ to find the optimal leaking memory for the reservoir.

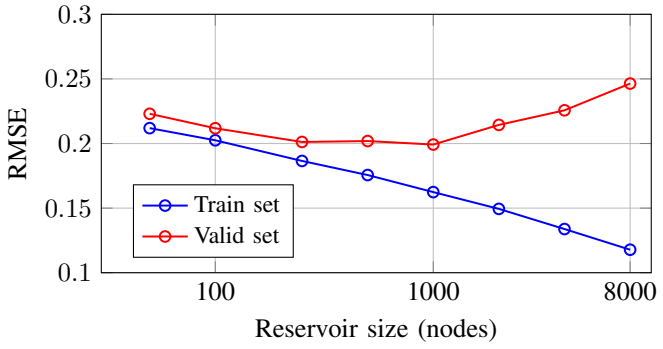

Fig. 11. Performance of RCN on the training and validation set as a function of reservoir size (number of nodes). A medium-size reservoir of 1000 nodes seems to be enough to capture the most information from the training set without getting overfitted.

starts to happen only when there are three times more model parameters compared to the training samples.

Although 200 ms has been empirically shown to be a suitable window for plasma behaviour monitoring, we also investigate the performance of RCN as a function of this prediction window. In Fig. 12 we compare the proposed RCN model with two other references: (1) Flat Base: which assumes that the signal at the intended future will be exactly the same as now, and (2) Lin Reg: which tries to predict the future by applying linear regression directly on the input features,i.e.,

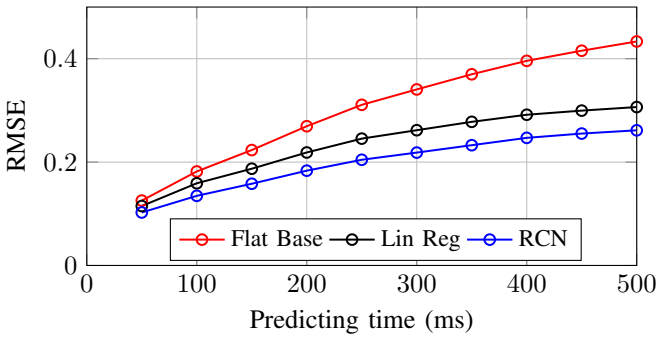

Fig. 12. RMSE of three models as a function of prediction window. RCN performs relatively 30% better than the Flat Baseline and also 15% better than the simple linear regression model.

bypassing the reservoir. This experiment show that on average the reservoir-based model performs relatively 30% better than the Flat Baseline and also 15% better than the simple linear regression model.

B. Training Time and Adaptation

An important asset of reservoir computing compared to state-of-the-art (deep) neural networks is its easy and fast training procedure. All the experiments in this work are conducted on a conventional CPU Intel® Core™ i7-3770.

Table II lists the time complexity of each of the training steps as a function of data and reservoir size along with actual times in seconds to train a 1000-node RCN on the given training data set.

Action	Complexity	Time (s)
Compute R & D $(*)$	$\mathcal{O}(N^{tr}\,N^{res})$	25
Update $\mathcal{R}$ & $\mathcal{D}$ $(*)$	$\mathcal{O}(N^{tr}\,(N^{res})^2)$	20
Compute $\mathbf{W}^{out}$	$\mathcal{O}((N^{res})^2\,N^{out})$	0.08

TABLE II
TIME COMPLEXITY AND ACTUAL TIMES (MEASURED ON AN INTEL® CORE™ I7-3770) FOR THE DIFFERENT STEPS OF THE FULL TRAINING OF A RESERVOIR WITH 1000 NEURONS USING 2,750 SHOTS (132,000 DATA POINTS). ITEMS TAGGED WITH $(*)$ CAN BE RUN IN PARALLEL.

Collecting $\mathcal{R}$ and $\mathcal{D}$ (lines 2 to 17 of Algorithm 1) for the whole training dataset only takes $45s$ ($160ms$ per shot) and training the output weights (line 19) is almost real-time ($80ms$).

This suggests that by storing and updating $\mathcal{R}$ and $\mathcal{D}$, we can almost instantly adapt the RCN model at the end of each shot, or even every few hundred milliseconds during a given shot.

Before investigating the adaptation ability of RCNs, we studied how much of the existing training data is in fact enough for prediction of the validation set. Therefore we define three scenarios for gradually increasing the size of the training set and evaluating the trained model on the validation set described in Section IV, (1) Sorted: the training and validation sets are derived from the original shot list, ordered chronologically. The first 70% is used for training and the model sees the training shots from the earliest to the

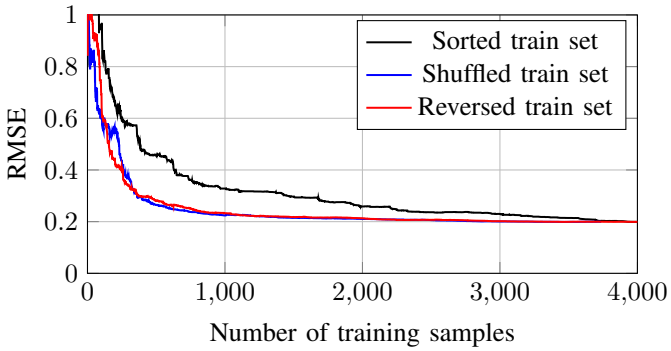

Fig. 13. Increasing number of training shots beginning from the oldest shot (Sorted), beginning from the most recent shot (Reversed) and picking the shots randomly (Shuffled).

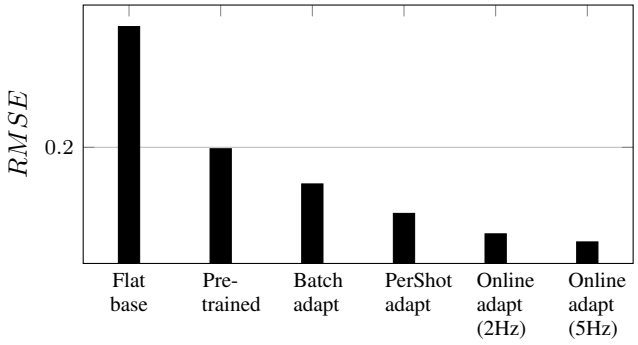

Fig. 14. Comparing the performance of three adaptation approaches on RCN: Batch adaptation, PerShot adaptation and Online adaptation. The latter has been studied with adaptation frequencies of 2Hz (every 500ms) and 5Hz (every 200ms).

most recent one. (2) Reversed: The same setup as above, but the model sees the training shots in time-reversed order. (3) Shuffled: all shots are shuffled and randomly assigned to the training and validation set. The comparison of these scenarios should give an indication of how many shots, and in which order, is optimal for a pre-trained RCN.

In Fig. 13, we gradually increase the number of training shots in these scenarios, retrain the model using Eq. 3 and evaluate the performance of the model on the whole validation set after each step. The comparison between the Sorted and Shuffled scenarios shows that it is not necessary to train the RCN with a very long history of shots. In fact, the model trained on the most recent 1000 shots performs equally well as a model that has been trained on the first 3000 available shots. This is important to avoid unnecessary complexity of the training procedure. Furthermore, the Shuffled scenario confirms that it is indeed redundant to train the model on all the available data. Only a quarter of the current data set contains almost all useful information to train the prediction model.

In order to study the adaptation capability of RCNs, we first train the model on the available training set and define three scenarios to adapt the model to the new environment which is the validation set: (1) Batch adaptation, in which the output weights are updated only after every batch of 10 shots, (2) PerShot adaptation, in which the output weights are updated after every shot of the validation set and (3) Online adaptation in which the model is adapted every 500 ms "during" the shot. In all scenarios, $\mathbf{W}^{out}$ is pre-trained by collecting $\mathcal{R}$ and $\mathcal{D}$ on the training data. During the adaptation, $\mathbf{W}^{out}$ is updated gradually by calculating $\mathcal{R}_A$ and $\mathcal{D}_A$ from the validation set and updating $\mathcal{R}$ and $\mathcal{D}$ (see Equation 5).

In Fig. 14 we evaluate these approaches and compare them with the non-adapted (pre-trained) RCN, as well as the Flat Baseline. As one can expect, increasing the frequency of adaptation improves the prediction performance of the model.

Table III lists the total adaptation time and the number of adaptations on the validation set for a 1000-node RCN based on the three aforementioned scenarios. The numbers show that even the online adaptation, which is the most intense scenario, can be accomplished fast enough on a conventional CPU.

	Batch adapt	PerShot adapt	Online adapt
Total adaptation time (s)	6	73	327
Number of adaptations	58	704	3392
Average time (ms)	103	104	96

TABLE III
TIME AND NUMBER OF ADAPTATIONS FOR THREE SCENARIOS OF ADAPTING A 1000-NODE RCN TO A VALIDATION SET OF 705 SHOTS.

In the aforementioned scenarios we assumed that there is a separated training data set to pre-train the RCN before evaluating and adapting the model under the test conditions. However, we also investigated how the RCN would behave in case we begin with a non-trained RCN and adapt it to the test environment as time evolves. Figure 15 shows the performance of three RCNs, namely, (1) pre-trained: the model is trained on the training set and evaluated on the test set, (2) pre-trained-adapted: the model is pre-trained on the training set and gets updated during the test, (3) not-trained-adapted: the model is trained on the test set gradually without any pre-training. This experiment shows that the pre-trained-adapted model outperforms the other two, especially compared to the pre-trained model towards the end of the test. Secondly, the not-trained model starts with poor prediction, as can be expected, but in the second half of the test (after around 250 shots) it gets close to the pre-trained-adapted model and in some examples it even outperforms the pre-trained model, which has been trained on 2,750 shots before testing.

As an example of the prediction of the proposed model, Fig. 16 depicts the five profiles in one particular shot from the test set. For each profile and at each timestep, we show the average over its 65 channels of target value, along with the predictions of the pre-trained RCN, PerShot and the Online adaptation. While the pre-trained RCN fails to predict the second half of the signals, which exhibits a significantly different pattern compared to the first half, adapting the model just before this shot slightly helps to improve the prediction. On the other hand, adapting the model every 500 ms significantly improves the prediction.

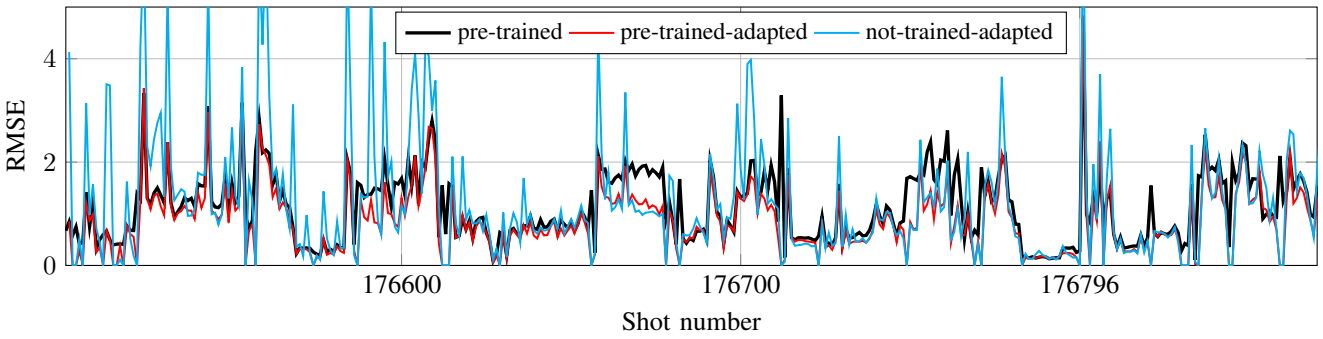

Fig. 15. Performance comparison of an RCN only trained on the training data (pre-trained), the same RCN getting adapted to the test data (pre-trained-adapted), and an RCN learning the task from scratch on the test shots.

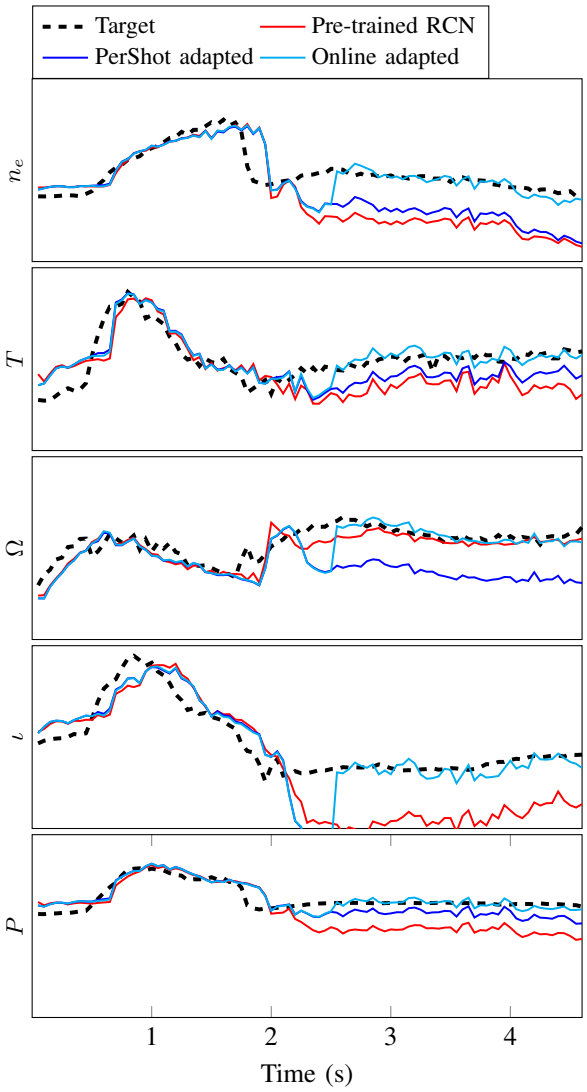

Fig. 16. Target and prediction of the five profiles for shot #176796. Each data-point at time t is the average of the 65-dimensional vector of the corresponding profile at that time.

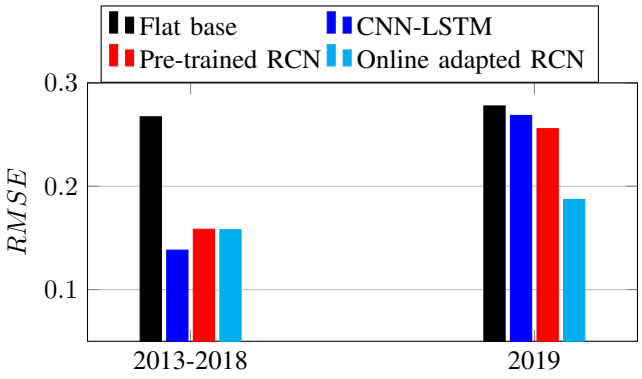

Fig. 17. Comparison of RCN and CNN-LSTM when training and testing are from the same pool (collected in 2013-2018) and when the test shots are from a different year (2019).

C. Comparing RCN with CNN-LSTM

Convolutional Neural Networks in combination with Long-Short Term Memory cells have increasingly become popular approaches for many data analysis tasks involving time series [8]. However, their complex training procedure demands a large amount of training data and time. Furthermore, these systems usually achieve promising performance when the training data is rich and large enough and testing environments share common properties with training. In this section we compare RCN with a well-configured CNN-LSTM framework [2] in two scenarios: (1) when training and testing shots are randomly selected from a pool of shots recorded between 2013 and 2018, and (2) when the test shots are collected from a completely different setup in the next year (2019). Figure 17 presents the performance of RCN and LSTM in these conditions. While LSTM achieves promising scores in the matched conditions, it drastically fails on the newly recorded shots. The pre-trained RCN shows the same behavior which likely means that there is a considerable difference between the new data and the training set. However the fast adaptation ability of RCN leads to 25% relative improvement. As far as the computational complexity is concerned, the training of the CNN-LSTM model took 5 hours on one NVIDIA V100 GPU and 8 IBM POWER9 CPU cores compared to 45 seconds for training the RCN on a Core i7 CPU.

[2]This code is also publicly available, on GitHub (https://github.com/jabbate7/plasma-profile-predictor)

VI. Conclusion and future work

Many complex machine learning paradigms, such as deep neural networks and CNNs require large amount of data and time to learn the task, hence, they are usually difficult to adapt to the new conditions. We have explored the potential of reservoir computing networks (RCNs) as interesting alternatives for data analysis and prediction tasks involving multidimensional time-series measured in complex physical systems. In this respect, the main assets of RCNs are their temporal information processing ability and yet very easy and fast training procedure.

In this study, we have addressed real-time profile prediction for condition monitoring in fusion devices using RCNs. We have laid special emphasis on model training complexity and adaptation under changing operational conditions, showing the possibility to update the model in a near real-time fashion. In this particular setting, every adaptation takes only 100 ms, which, for instance, can be compared with the typical suggested prediction time of about 200ms for disruption avoidance in tokamaks.

Moreover, we have explored the main hyperparameters of RCNs and studied their impact on the activation and memory of the reservoir. For instance, our experiments have shown that the input and recurrent connections of the model can be very sparse without compromising the model performance, which is useful for model storage. We have also shown that the training of model can be accomplished in parallel. Thus, RCNs are suitable for pattern recognition in big data.

As a follow up to this work, we plan to utilize larger data sets covering a greater diversity of plasma conditions. In particular, although usually this is not how fusion devices are operated, it would be extremely useful to obtain more data from shots with semi-random changes to actuators during the current flattop.

Furthermore, it would be interesting to investigate more complex architectures of RCNs, such as multilayer models, as well as more sophisticated topologies in connecting the reservoir neurons. For instance, a more structured input to reservoir connections would divide the reservoir neurons into segments, where each area learns specific properties of the patterns in the training data. This might lead to better understanding and control of the reservoir activation.

Finally and although in this paper we focused on plasma profile prediction as a use-case, we believe that our findings can be easily generalized to other applications and across several disciplines.

Acknowledgment

Part of data analysis is performed using the OMFIT integrated modeling framework. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Fusion Energy Sciences, DOE ARPA-E, using the DIII-D National Fusion Facility, a DOE Office of Science user facility, under Awards DC-AC02-09CH11466, DE-SC0015480, DE-SC0015878, DE-FC02-04ER54698, DE-AR0001166 and Field Work Proposal No. 1903. A. Jalalvand would like to thank the Special Research Fund of Ghent University for funding his research (BOF19/PDO/134).

Disclaimer

References

[1] F. C. Schuller, "Disruptions in tokamaks," *Plasma Physics and Controlled Fusion*, vol. 37, no. 11A, pp. A135–A162, nov 1995. [Online]. Available: https://doi.org/10.1088%2F0741-3335%2F37%2F11a%2F009

[2] P. de Vries, M. Johnson, B. Alper, P. Buratti, T. Hender, H. Koslowski, and V. R. and, "Survey of disruption causes at JET," *Nuclear Fusion*, vol. 51, no. 5, p. 053018, apr 2011. [Online]. Available: https://doi.org/10.1088%2F0029-5515%2F51%2F5%2F053018

[3] J. Hernandez, A. Vannucci, T. Tajima, Z. Lin, W. Horton, and S. McCool, "Neural network prediction of some classes of tokamak disruptions," *Nuclear fusion*, vol. 36, no. 8, p. 1009, 1996.

[4] D. Wroblewski, G. Jahns, and J. Leuer, "Tokamak disruption alarm based on a neural network model of the high-beta limit," *Nuclear Fusion*, vol. 37, no. 6, p. 725, 1997.

[5] G. Pautasso, S. Egorov, C. Tichmann, J. Fuchs, A. Herrmann, M. Maraschek, F. Mast, V. Mertens, I. Perchermeier, C. Windsor *et al.*, "Prediction and mitigation of disruptions in ASDEX upgrade," *Journal of nuclear materials*, vol. 290, pp. 1045–1051, 2001.

[6] C. G. Windsor, G. Pautasso, C. Tichmann, R. J. Buttery, and T. C. Hender, "A cross-tokamak neural network disruption predictor for the jet and asdex upgrade tokamaks," *Nucl. Fusion*, vol. 45, p. 337, 2005.

[7] B. Cannas, A. Fanni, P. Sonato, M. Zedda, J.-E. contributors *et al.*, "A prediction tool for real-time application in the disruption protection system at JET," *Nuclear Fusion*, vol. 47, no. 11, p. 1559, 2007.

[8] J. Kates-Harbeck, A. Svyatkovskiy, and W. Tang, "Predicting disruptive instabilities in controlled fusion plasmas through deep learning," *Nature*, vol. 568, no. 7753, pp. 526–531, 2019. [Online]. Available: http://dx.doi.org/10.1038/s41586-019-1116-4

[9] R. J. Buttery, T. C. Hender, D. F. Howell, R. L. Haye, S. Parris, O. Sauter, and C. G. Windsor, "On the form of ntm onset scalings," *Nucl. Fusion*, vol. 44, p. 678, 2004.

[10] G. A. Rattá, J. Vega, A. Murari, G. Vagliasindi, M. F. Johnson, and P. C. D. Vries, "An advanced disruption predictor for jet tested in a simulated real-time environment," *Nucl. Fusion*, vol. 50, p. 025005, 2010.

[11] J. Vega, S. Dormido-Canto, J. M. López, A. Murari, M. J. Ramírez, R. Moreno, M. Ruiz, D. Alves, and R. Felton, "Results of the jet real-time disruption predictor in the iter-like wall campaigns," *Fusion Eng. Des.*, vol. 88, p. 1228, 2013.

[12] Y. Zhang, G. Pautasso, O. Kardaun, G. Tardini, and X. D. Zhang, "Prediction of disruptions on asdex upgrade using discriminant analysis," *Nucl. Fusion*, vol. 51, p. 063039, 2011.

[13] C. Rea, R. Granetz, K. J. Montes, R. A. Tinguely, N. W. Eidietis, J. M. Hanson, and B. S. Sammuli, "Disruption prediction investigations using machine learning tools on diii-d and alcator c-mod," *Plasma Phys. Controlled Fusion*, vol. 60, p. 084004, 2018.

[14] K. Montes, C. Rea, R. Granetz, R. Tinguely, N. Eidietis, O. Meneghini, D. Chen, B. Shen, B. Xiao, K. Erickson, and M. Boyer, "Machine learning for disruption warnings on alcator c-mod, DIII-d, and EAST," *Nuclear Fusion*, vol. 59, no. 9, p. 096015, jul 2019. [Online]. Available: https://doi.org/10.1088\%2F1741-4326\%2Fab1df4

[15] Y. Fu, D. Eldon, K. Erickson, K. Kleijwegt, L. Lupin-Jimenez, M. D. Boyer, N. Eidietis, N. Barbour, O. Izacard, and E. Kolemen, "Machine learning control for disruption and tearing mode avoidance," *Physics of Plasmas*, vol. 27, no. 2, p. 022501, 2020. [Online]. Available: https://doi.org/10.1063/1.5125581

[16] H. Jaeger, "Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the echo state network approach," GMD Report 159, German National Research Center for Information Technology, Tech. Rep., 2002.

[17] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Networks*, vol. 115, pp. 100 – 123, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608019300784

[18] C. Gallicchio and A. Micheli, "Richness of deep echo state network dynamics," in *Advances in Computational Intelligence*, I. Rojas, G. Joya, and A. Catala, Eds. Cham: Springer International Publishing, 2019, pp. 480–491.

[19] A. Jalalvand, F. Triefenbach, K. Demuynck, and J.-P. Martens, "Robust continuous digit recognition using reservoir computing," *Computer Speech and Language*, vol. 30, no. 1, pp. 135 – 158, 2015.

[20] M. Xu, P. Baraldi, S. Al-Dahidi, and E. Zio, "Fault prognostics by an ensemble of echo state networks in presence of event based measurements," *Engineering Applications of Artificial Intelligence*, vol. 87, p. 103346, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197619302854

[21] C. Gallicchio, A. Micheli, and L. Pedrelli, "Design of deep echo state networks," *Neural Networks*, vol. 108, pp. 33 – 47, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608018302223

[22] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, "Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach," *Physical Review Letters*, vol. 120, p. 024102, Jan 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.120.024102

[23] A. Jalalvand, B. Vandersmissen, W. De Neve, and E. Mannens, "Radar signal processing for human identification by means of reservoir computing networks," in *IEEE Radar Conference (RadarConf)*, 2019, pp. 1–6.

[24] H. Jaeger, "The ''echo state'' approach to analysing and training recurrent neural networks - with an erratum note," GMD Report 148, German National Research Center for Information Technology, Tech. Rep., 2001. [Online]. Available: http://www.faculty.jacobs-university.de/hjaeger/pubs/EchoStatesTechRep.pdf

[25] A. Jalalvand, K. Demuynck, and J.-P. Martens, "Noise robust continuous digit recognition with reservoir-based acoustic models," in *Proc. ISPACS*, 2013, p. ID:99.

[26] D. Verstraeten, "Reservoir computing: computation with dynamical systems," Ph.D. dissertation, Ghent University, 2009.

[27] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural Computation*, vol. 7, no. 1, pp. 108–116, Jan 1994.

[28] D. W. Marquardt and R. D. Snee, "Ridge regression in practice," *The American Statistician*, vol. 29, no. 1, pp. 3–20, 1975.

[29] R. Penrose, "A generalized inverse for matrices," *Proc. The Cambridge Philosophical Society*, vol. 51, pp. 406–413, Jul. 1955.

[30] J. a. Gama, I. Žliobaitundefined, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, Mar. 2014. [Online]. Available: https://doi.org/10.1145/2523813

[31] C. Gallicchio, "Sparsity in reservoir computing neural networks," 2020.

[32] A. Jalalvand, W. D. Neve, R. V. de Walle, and J. P. Martens, "Towards using reservoir computing networks for noise-robust image recognition," in *Proc. IJCNN*, July 2016, pp. 1666–1672.

[33] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Trans. Syst., Man, Cybern. B*, vol. 42, no. 2, pp. 513–529, April 2012.

[34] M. Lukoševičius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7700, pp. 659–686.

[35] J. Stillerman, T. Fredian, K. Klare, and G. Manduchi, "Mdsplus data acquisition system," *Review of Scientific Instruments*, vol. 68, pp. 939 – 942, 02 1997.

[36] O. Meneghini, S. Smith, L. Lao, O. Izacard, Q. Ren, J. Park, J. Candy, Z. Wang, C. Luna, V. Izzo, B. Grierson, P. Snyder, C. Holland, J. Penna, G. Lu, P. Raum, A. McCubbin, D. Orlov, E. Belli, N. Ferraro, R. Prater, T. Osborne, A. Turnbull, and G. Staebler, "Integrated modeling applications for tokamak experiments with omfit," *Nuclear Fusion*, vol. 55, no. 8, p. 083008, 2015. [Online]. Available: http://iopscience.iop.org/article/10.1088/0029-5515/55/8/083008/meta