ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-THPAB201

A MACHINE LEARNING TECHNIQUE FOR DYNAMIC APERTURE COMPUTATION

M. Ben Ghali, B. Dalena*
CEA, Irfu, DACM, Paris-Saclay University, Gif-sur-Yvette, France

Abstract

Currently, dynamic aperture calculations of high-energy hadron colliders are performed through computer simulations, which are both a resource-heavy and time-costly processes. The aim of this study is to use a reservoir computing machine learning model in order to achieve a faster extrapolation of dynamic aperture values. A recurrent echo-state network (ESN) architecture is used as a basis for this work. Recurrent networks are better fitted to extrapolation tasks while the reservoir echo-state structure is computationally effective. Model training and validation is conducted on a set of "seeds"corresponding to the simulation results of different machine configurations. Adjustments in the model architecture, manual metric and data selection, hyper-parameters tuning and the introduction of new parameters enabled the model to reliably achieve good performance on examining testing sets.

INPUT DATA

The Dynamic Aperture (DA) represents the region of stable motion of a particle after a certain number of revolutions in an accelerator [1]. The sources of the instability are magnetic fields and elements placement imperfections. In large hadron machines , since these magnetic field errors are not known precisely, Monte Carlo simulations of different machine configurations using Gaussian distributed errors values are usually performed [2,3]. DA is typically used to: i) define tolerances on field quality; ii) define non-linear corrections schemes in the design phase of the accelerator; iii) verify the effect of correction in real running machines. The Dynamic Aperture values used in this paper have been estimated using the initial amplitude corresponding to particle lost after the 10^n revolutions. Alternatively, the divergence of nearby trajectories in the phase space is followed to define the limit between chaotic and non-chaotic motion (Lyapounov exponent) [4]. However, the methods listed above require both massive tracking simulations (from one day to one week or more depending on the test case and its complexity). For example, in order to determine the amplitude at which the particles are stable after 10^5 revolution, we track 60 particles with initial amplitudes distributed over 59 angles in the $x - y$ phase space, up to a maximum amplitude of 25 σ (i.e. number of beam size) in steps of 1 σ for each machine configuration (called seeds), see Fig. 1. These tracking simulations can be executed in parallel on a cluster or on the BOINC platform [5, 6].

Due to its computational effort, DA is presently computed at typically 10^5-10^6 number of revolutions in the accelerator, which is much less (~90 s) than the LHC beam lifetime (up to 12 hours). Starting from the ensemble of initial amplitude of particles lost in the $x - y$ phase space (or equivalently the maximum initial amplitude of stable particles), shown in Fig. 1, DA can be defined, according to Ref. [2], as:

$$DA(N) = \frac{2}{\pi} \int_0^{\pi/2} r_s(\theta; N) d\theta, \quad (1)$$

where N is the number of revolutions of the particle in the accelerator (called turns), r_s is the last stable amplitude (disregarding stability islands non-connected to the origin) and θ is the angle in the $x - y$ phase space. Thus, a value of DA can be calculated as a function of turns, which is shown in Fig. 2 for one configuration of the machine. The numerical DA values, coming from the tracking simulations, are sparse in the number of turns. In order to have data at regular turns interval we have, as first approximation, used the fill-forward methods of pandas library. This can allow to get DA value at each turn, but to reduce computational cost without loosing real numerical values coming from the numerical simulations, one value each 100 turns is used in the following. In Ref. [7], Gaussian processes are used to generate points in between the DA numerical values.

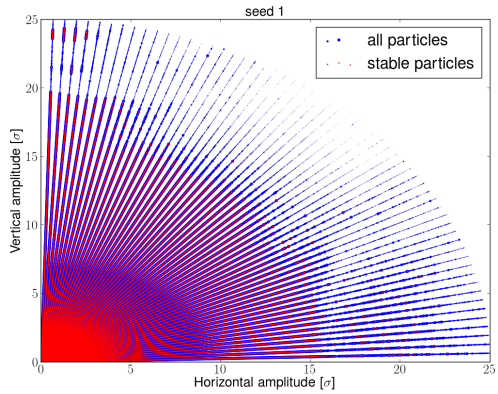

Figure 1: Stable initial amplitude of particles (normalized to beam sizes) after 10^5 revolution (red) and all initial amplitude of particles simulated (blue). We greatly acknowledge all BOINC volunteers who supported LHC@Home project, giving for free their CPU time and allowing these results to be produced.

ECHO STATE NETWORK

Historically, while able to efficiently perform interpolation and classification tasks, machine learning models have notoriously struggled with extrapolation. Since we are aiming

* barbara.dalena@cea.fr

ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-THPAB201

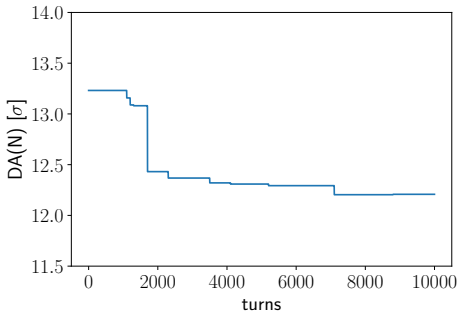

Figure 2: Dynamic Aperture as a function of number of revolutions in the accelerator (turns) for one machine configuration.

to extrapolate functions using the smallest possible number of known data points, as points are currently simulated and thus too costly to compute, this makes Artificial Neural Network (ANN) not suitable for our purpose. Therefore, we explore the possibility to predict DA with a generative Recurrent Neural Networks (RNN). The latter are feed-forward structures which can keep track of previous entries and their outcomes, and make future predictions. Particularly, we use Echo State Networks (ESN) [8, 9], whose general scheme is shown in Fig. 3. Instead of training our neural network with parts of a given target function as it is usually the case in interpolation machine learning problems, the network is trained over a number of sample data sets each corresponding to a fully simulated DA computation. This is meant for the network to learn the inherent behaviour of the DA as function of turns. The trained network is then fed with the initial data points for a new target function and is run in generative mode to predict the remaining points for that specific instance. ESN are a family of recurrent neural networks which are comprised of two main components:

- The "reservoir", a nonlinear transformation applied to all inputs. It depends on the $\mathbf{W_{in}}$ and $\mathbf{W}$ matrices which are generated randomly at each new instance of the network (with respect to certain criteria), and never changed during training or prediction.
- The "readout", the final layer that computes the output, given the input transformed by the reservoir. This is the only trainable part of an ESN and is optimized during training. In our case the readout layer is a linear combination of reservoir elements. This combination is defined by $\mathbf{W_{out}}$ matrix which is computed via linear regression at the end of the training phase using all training data sets.

The advantage of this kind of network is the faster computation time. Even with more complex readout layers it remains a relatively simple output to train, also because the reservoir, sparse by definition, is not computationally-heavy.

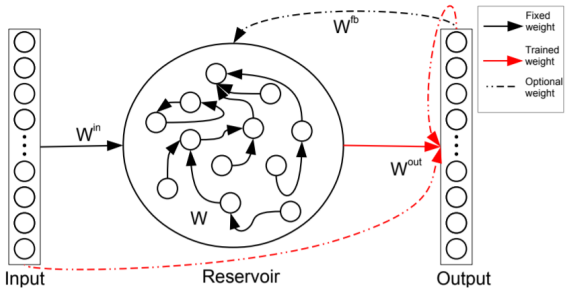

Figure 3: Echo State Network general scheme.

STRUCTURE OF MODEL AND HYPER-PARAMETERS

Here we describe more in the details our ESN implementation and the tuning of its hyper-parameters. First, $\mathbf{W_{in}}$ and $\mathbf{W}$ (which are used to update the reservoir) are generated randomly with the sparse condition respected. $\mathbf{W}$ must also be of a spectral radius smaller than 1, which is the condition for the algorithm to converge (called the "reservoir condition"). This is easily achieved by dividing the randomly generated matrix's values by its greatest eigenvalue. At each iteration, given an input $\mathbf{u}$ (in our case the DA as function of turns), the reservoir state $\mathbf{x}$ is updated according to the following formula:

$$\mathbf{x}(n+1) = (1-a)\,\mathbf{x}(n) + t\,tanh(\mathbf{W_{in}}[1, \mathbf{u}(n+1)] + \mathbf{W}\mathbf{x}(n))$$

, where a is the reservoir *leaking rate*, t is the *tanh leaking rate* and [1,.] is the stack of a constant and the input vector. $tanh()$ with a vector argument is defined as the vector whose components are the hyperbolic tangents of the corresponding components of the argument vector. It acts as the *activation function* of the network and ensures the non-linearity of the reservoir transformation. The input vector, $\mathbf{u}$, will depend on whether the network is running in training or generative mode. The reservoir output is obtained as:

$$\mathbf{y}(n) = g(\mathbf{W_{out}}[1, \mathbf{u}(n), \mathbf{x}(n)]),$$

where $\mathbf{W_{out}}$ is randomly initiated then computed through training, and g is the readout activation function. In the case discussed in the present paper g is the identity, keeping the linearity of the output layer. Studies of non-linear readout layer are in progress.

The hyper-parameters of the presented model are:

- **Reservoir size**: the dimension of the state $\mathbf{x}$;
- **Leaking rate** a;
- **Reservoir connectivity**: the percentage of non-zero connections in the reservoir. This dictates how sparse the reservoir matrix is by setting a number of reservoir weights to 0. This number is a ratio than can go from 0 to 1 where 1 corresponds to the default randomly-generated weights and 0 to a completely sparse reservoir;

ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-THPAB201

- **Spectral radius**: is the largest eigenvalue of the reservoir matrix. The reservoir weights are scaled by their biggest eigenvalue, and then multiplying them by our "spectral radius"parameter.
- **Leaking rate** t: an additional leaking rate parameter which complements the previous one, a;
- **Teacher-forcing**: consists in the introduction of a feedback loop in the reservoir update. It has a slightly positive impact on the performance.
- Other parameters: **input scaling** (scaling of input weights), **noise level** (random noise term in the reservoir update), **n_drop** (the number of reservoir states to drop in the beginning before starting storing) are either unused or set to "default"values (where they do not intervene in the network).

The tuning of hyper-parameters was mostly done through a grid search script. We use the Median Absolute Error (MEA) to establish the performance of the ESN but further systematic studies are foreseen. We looked for correlation in reservoir size and connectivity, spectral radius and connectivity, and between the two leaking rates (a and t). None of them has shown a sharp correlation. While there seems to be a general decrease in error as we use bigger reservoirs, we can still easily achieve the best performance with smaller ones, if we select the right connectivity values. This is relevant as reservoir size heavily affects the computation time. Overall, connectivity seems to have less impact on error values than spectral radius. There seems to be a 1:1 correlation between the two leaking rates (t and a) but with a large base. This leads to a preference of small but not equals values for both. In particular, too small tanh coefficient (t) values result in a stationary reservoir which does not update. Finally, we have tested the linear readout layer. Ridge regression is traditionally used in the readout layer to fit data. It is defined by the addition of an L2 norm penalty term (λ). Opposed to this is the Lasso regression which uses an L1 norm penalty. In the Elastic-Net regression [10], using the alpha parameter (L1 rate), we can give greater weight to the Ridge or Lasso component. Our test confirms that a Ridge regression is the best fit for our model.

PERFORMANCE

Using the best grid search results for the hyper-parameters (spectral radius : 0.35; connectivity : 0.8; leaking rate : 0.25; tanh leak : 1; Ridge coefficient : 5.0; Reservoir size : 250), (-1,1) MinMax Scaling of the input data, teacher-forcing, 100-step data sampling, and 20:39 training to testing seed split, the model is able to predict the DA values. It shows a capacity of yielding non-linear solution which well describes the data of several seeds used for testing, as shown in the top panel of Fig. 4.

The Median absolute error scores are within the precision margin for DA applications for most of the seeds. It does however come with the drawbacks of a few outliers seeds, where predictions are worse, as can be seen in the bottom panel of Fig. 4. When we run the model on the longer 100000-turn seeds we immediately see a drop in performance. Not only we have higher error values, we also see an increase in the number of divergent seeds. This may suggests that the number of parameters used for training might not be enough for the longer turns data. Finally, we retry predicting without the use of training seeds by training the model on the start of each data set. We find that the results are unreliable as model diverges in a seemingly completely random fashion.

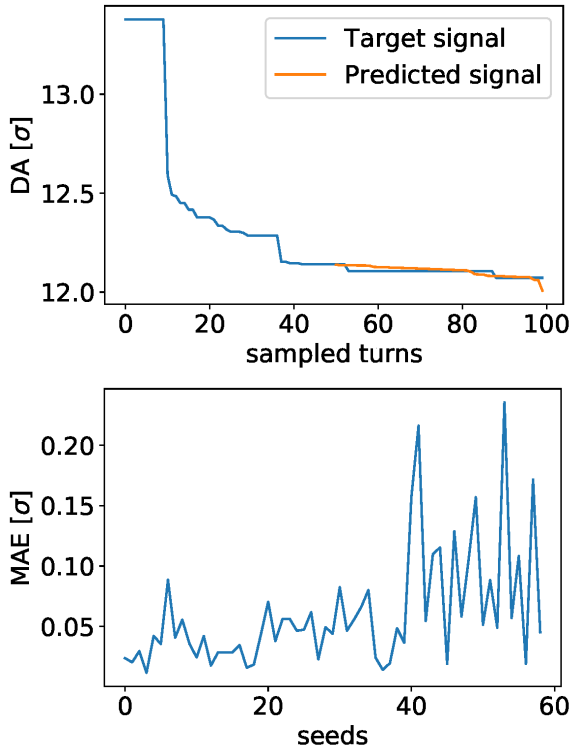

Figure 4: Dynamic Aperture as a function of sampled turns (each 100 turns) compared with one of the best Echo State Network prediction (top panel); Median Absolute Error vs seed (bottom panel) for training (up to 20) and test data.

CONCLUSION AND PERSPECTIVES

The Echo State Network implemented in this work shows strong potential to predict the time evolution of dynamic aperture. However, its capacity in reproducing test data requires several complete simulations to be used for training. A model which does not need as much data as current examples must be achieved. To this purpose both the possibility of using several reservoirs (randomly initialized) and to use alternative readout layers are foreseen.

REFERENCES

[1] E. Todesco *et al.*, "Dynamic aperture estimates and phase-space distortions in nonlinear betatron motion", *Phys. Rev. E*,vol. 53, p. 4067, 1996. doi:10.1103/PhysRevE.53.4067

ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-THPAB201

[2] E. H. Maclean *et al.*, "Innovative method to measure the extend of the stable phase-space region of proton synchrotrons", *Physical Review Accelerators and Beams*, vol. 22, p. 034002, 2019. doi:10.1103/PhysRevAccelBeams.22.034002

[3] M. Giovannozzi, "Proposed scaling law for intensity evolution in hadron storage rings based on dynamic aperture variation with time", *Phys. Rev. ST Accel. Beams*, vol. 15, p. 024001, 2012. doi:10.1103/PhysRevSTAB.15.024001

[4] F. Schmidt, F. Willeke, and F. Zimmermann, "Comparison of methods to determine long-term stability in proton storage rings", *Particle Accelerators*, vol. 35, pp. 249-256, 1991.

[5] D. P. Anderson, "BOINC: a System for Public-Resource Computing and Storage", in *Proc. of the 5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, 2004, pp. 4-10. doi:10.1109/GRID.2004.14

[6] J. Barranco *et al.*, "LHC@Home: a BOINC-based volunteer computing infrastructure for physics studies at CERN", *Open Engineering*, vol. 7, p. 378, 2017. doi:10.1515/eng-2017-0042

[7] M. Giovannozzi *et al.*, "Machine Learning Applied to the Analysis of Nonlinear Beam Dynamics Simulations for the CERN Large Hadron Collider and Its Luminosity Upgrade", *Information*, vol. 12, p. 53, 2021. doi:10.3390/info12020053

[8] H. Jaeger, "The echo state approach to analysing and training recurrent neural networks", German National Research Institute for Computer Science, Germany, Rep. GMD-Report 148, 2001.

[9] J. Pathak *et al.*, "Using Machine learning to replicate chaotic attractors, and calculate lyapunov exponent from data", *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 12, p. 121102, Dec. 2017. doi:10.1063/1.5010300

[10] Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python", *JMLR*, vol. 12, pp. 2825-2830, 2011. arXiv:1201.0490