Fusion Engineering and Design 163 (2021) 112125

Contents lists available at ScienceDirect

Fusion Engineering and Design

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

Accelerated version of NUBEAM capabilities in DIII-D using neural networks

Shira M. Morosohk [a,*], Mark D. Boyer [b], Eugenio Schuster [a]

[a] Lehigh University, Bethlehem, PA 18015, USA
[b] Princeton Plasma Physics Laboratory, Princeton, NJ 08540, USA

ARTICLE INFO

Keywords:
Machine learning
Neural networks
NUBEAM

ABSTRACT

A neural network model of the effects of neutral beam injection on DIII-D has been developed. The training and testing data used by the model have been generated by the NUBEAM module of TRANSP for experimental discharges from the 2018 DIII-D campaign. Using a principle component analysis to reduce the dimensionality of profile data, the model has been shown to reproduce the results of the Monte Carlo code NUBEAM with a high level of accuracy and an execution time orders of magnitude faster than the execution time of NUBEAM. This makes the neural network model uniquely suited to applications in model-based scenario planning (off-line) and active control (on-line), where a large number of simulation runs are required by the associated optimization tasks that need to be performed before and during the discharge.

1. Introduction

In order to maintain stability and maximize performance in tokamak plasmas, the spatial distributions of densities, temperatures, current, and momentum, among other factors, must be carefully controlled. The evolution of these profiles is described by a system of nonlinear partial differential equations, which in many applications cannot be modeled from first principles due to the extreme calculation power that would be required for their solution. Reduced physics-based models such as TRANSP [1,2] are commonly used instead in both analytic and predictive capacities. However, this type of physics-oriented model is still too time-consuming to be useful for certain applications such as between-shots discharge planning and real-time control. Because of this, a different set of control-oriented models with significantly faster calculation times is needed. Fast models of the plasma response to different types of actuation, such as neutral beam injection, are especially necessary for model-based control design. One approach to developing control-oriented models is to use empirical scaling laws [3]. These empirical models can easily achieve a calculation time fast enough to be useful for control applications, but may only be valid for specific plasma scenarios, which can lead to a considerable decrease in accuracy as plasma operation deviates from the reference scenario.

Another approach to building control-oriented models is to train machine learning algorithms such as neural networks [4] to reproduce the function of interest. Neural networks are based on the idea of building a mathematical model that operates in a similar way to the nervous system in order to reproduce some function. They have been proven to be capable of learning any function, no matter how complicated or nonlinear, given adequate training data and at least one hidden layer with enough neurons in that layer [5]. Each neuron in the network receives input signals from other neurons, processes them, and outputs a different signal. In a multi-layer perceptron (MLP) neural network, as illustrated in Fig. 1, multiple neurons are arranged in layers, and the signals move in a single direction from the inputs to the outputs. In this diagram, x_1, x_2, and x_3 make up the input layer, f^1 and f^2 make up the output layer, and in between there is one hidden layer. Each neuron in the hidden and output layers has a value determined by its inputs, the weights of each connection learned during training, and its activation function. Activation functions introduce a nonlinear component to the neural network calculation, which allows the network to learn nonlinear functions. Typical activation functions used for a regression problem are the rectified linear unit (ReLU), which is equal to zero for negative inputs and equal to the input for positive inputs, for neurons in the hidden layers. For neurons in the output layer, a linear activation function is used. When a neural network is initially created, the weights associated with each connection between neurons are randomized. During the training process, the network is given a set of inputs with known outputs. For each input in the training set, the network predicts an output

* Corresponding author.
E-mail address: morosohk@lehigh.edu (S.M. Morosohk).

https://doi.org/10.1016/j.fusengdes.2020.112125
Received 3 August 2020; Received in revised form 18 November 2020; Accepted 19 November 2020
Available online 9 December 2020

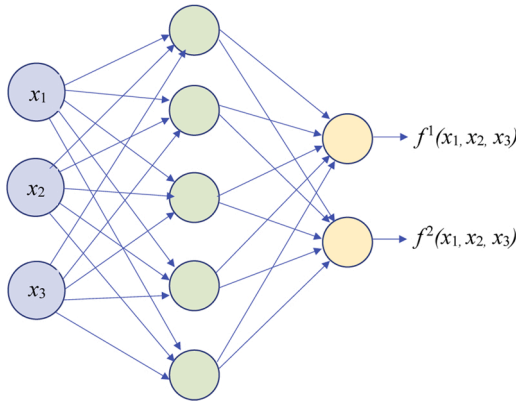


Fig. 1. Structure of a multi-layer perceptron neural network [6].

and compares it to the correct output using a loss function chosen by the developer which quantifies the error between the prediction and the true output. The network tracks the trend in the loss function between each update of the weights to determine if the training is moving in the right direction. Loss functions can be as simple as a mean squared error between the predicted and true outputs, or can be significantly more complicated and tuned to the specific system. A gradient descent algorithm is then used to update the weights with the goal of minimizing the loss function. Training ends when the value of the loss function is satisfactorily low.

Recently, neural networks have been created to replicate the results of a number of physics-based plasma models, achieving significantly reduced calculation time without major sacrifices in accuracy [7,8]. These neural-network models can then be integrated into predictive codes [9] to evaluate the state of the plasma much faster that could be done using the original models. Specifically, a neural network version of the TRANSP Monte Carlo neutral beam module NUBEAM [10,11], referred to as NubeamNet, was developed for the National Spherical Torus eXperiment Upgrade (NSTX-U) [12]. The neural-network model can predict the effects of neutral beam injection on the plasma in a fraction of the time NUBEAM requires, with similar levels of accuracy. This work aims to continue along these lines by creating a real-time capable version of NUBEAM which is valid for DIII-D. The DIII-D neutral beam system contains eight independently modulated neutral beam sources that can provide a combined power of up to approximately 20 MW. In TRANSP, each one of the powers associated with the two 150° off-axis beams in DIII-D is divided into four different components in order to account for different tilting configurations. Because of this, NUBEAM uses fourteen NBI power inputs instead of the eight physical neutral beam powers. Therefore, the neural network developed in this work also uses these fourteen powers as inputs. It is anticipated that this model will aid in optimal scenario planning and iterative control algorithm design, as well as other control applications including real-time control, estimation and forecasting in DIII-D [13–15].

Reduced analytical models [16] do exist as an alternative to this neural network model. However, reduced analytical models may require more significant assumptions to be made about the system, such as taking into account finite-orbit-width effects using an orbit average of the beam deposition. The neural network approach does not require any simplifying assumptions to be made beyond those used in NUBEAM. Instead it relies entirely on data, and is only valid for inputs within the range of its training data. Moreover, available analytical models do not calculate torque, which would be needed for certain model-based control applications such as rotation control.

This paper is organized as follows. In Section 2, the development of the dataset used to train and evaluate the model is described. In Section 3, the method used to determine the topology of the model is explained. In Section 4, predictions are shown using data that was not used in the training or parameter tuning stages. In Section 5, conclusions and plans for future work are discussed.

Table 1
Inputs of the neural network model.

Inputs		
Symbol	Name	Units
Z_{eff}	Mean effective charge	
$n_{0,out}$	Edge neutral density	cm^{-3}
R_0	Major radius	m
κ	Elongation	
I_p	Plasma current	A
a	Minor radius	m
$B_{\phi,v}R$	Vacuum toroidal field	T cm
δ_u	Upper triangularity	
δ_l	Lower triangularity	
$P_{inj}01-14$	Injected power for each beam	W
T_e	Electron temperature (profile)	eV
n_e	Electron density (profile)	cm^{-3}
q	Safety factor (profile)	
D_f	Anomolous fast ion diffusivity (profile)	cm^2/s

2. Dataset development

One of the advantages of using neural networks to model a system as opposed to highly reduced physics-based models is that neural networks make no assumptions about the structure of the model. Given sufficient training data, they are able to learn and reproduce highly nonlinear relationships. However, due to their complete reliance on data, they are unable to extrapolate for inputs outside of the range seen during the training process. While neural networks are incredibly capable of producing accurate results within the range of their training data, the outputs of any calculation for which the inputs are far from any inputs in the training dataset may be far from accurate and often non-physical. To account for this, it was decided that this neural network model would focus only on the range of inputs that are realistic for DIII-D. To that end, the training dataset was based on 83 shots from the 2018 DIII-D campaign. The dataset was then augmented by calling roughly 1000 TRANSP runs based on those shots and varying key inputs, for a total of 200,153 time slices in the dataset. The mean effective charge (Z_{eff}) was varied uniformly between 1.5 and 5, while the edge neutral density ($n_{0,\text{out}}$) was varied uniformly between 5×10^{10} and $1\times10^{13.5}$. The anomolous fast ion diffusivity was assumed to have the form

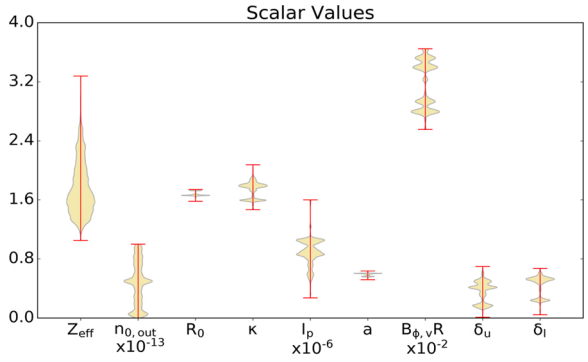
$$D_f(\widehat{\rho}) = D_{f,1} + (D_{f,0} - D_{f,1})(1-\widehat{\rho}^{\alpha_f})^{\beta_f}. \quad (1)$$

Each run used either the classical ($D_{f,0} = D_{f,1} = 0$), flat ($D_{f,0} = D_{f,1} = D_{f,\text{mag}}$), or peaked ($D_{f,1} = D_{f,\text{mag}}, D_{f,0} = 0, \alpha_f = 2, \beta_f = 4$) spatial profiles. These three shapes of the D_f profile are commonly used in TRANSP analyses in an effort to make NUBEAM outputs consistent with experimental data, so including these three shapes in the training data provides the network with the capability of predicting a wide range of plasma scenarios. The value of $D_{f,\text{mag}}$ was varied uniformly between 1 and 50,000 cm^2/s. In addition, the fidelity of NUBEAM was increased to 15,000 particles for each TRANSP run to increase the smoothness of the profile outputs.

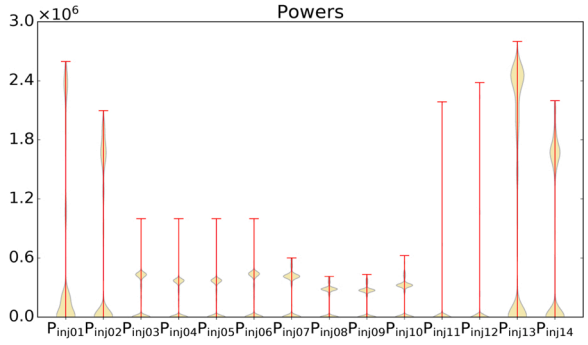
Of the set of 1000 TRANSP runs, 80% were randomly assigned to the training set, 10% were assigned to the testing set to be used in evaluating the final model, and the remaining 10% were assigned to the validation set to be used in parameter tuning. Parameter tuning is the process of determining the optimal values of the manually assigned network parameters, which are known as hyperparameters (see Section 3). Separating the data in this way allows the model to be tested on data that it has not seen during the training process, and therefore ensures that the

Table 2
Outputs of the neural network model.

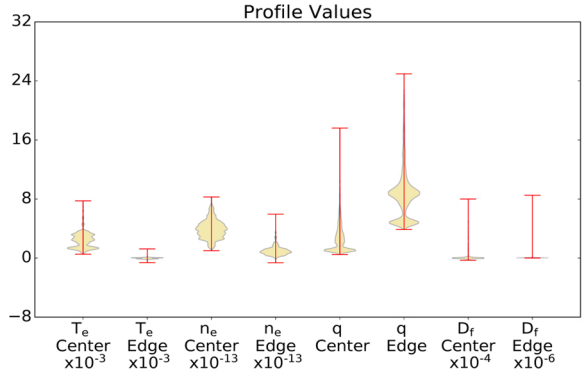
Outputs Symbol	Name	Units
s_{neut}	Total neutron rate	s^{-1}
P_{shine}	Shine-through power	W
P_{cx}	Charge-exchange power loss	W
P_{orb}	Orbit power loss	W
$P_{b,e}$	Beam heating to electrons (profile)	W/cm^3
$P_{b,i}$	Beam heating to ions (profile)	W/cm^3
$T_{b,e}$	Beam torque to electrons (profile)	Nm/cm^3
$T_{b,i}$	Beam torque to ions (profile)	Nm/cm^3
n_b	Beam ion density (profile)	cm^{-3}
j_b	Beam current drive (profile)	A/cm^2
p_{fast}	Fast ion pressure (profile)	Pa



(a) Range of training data for scalar inputs.



(b) Range of training data for neutral beam power inputs.



(c) Range of training data for profile inputs at the center and edge of the plasma.

Fig. 2. Range of inputs in the training dataset.

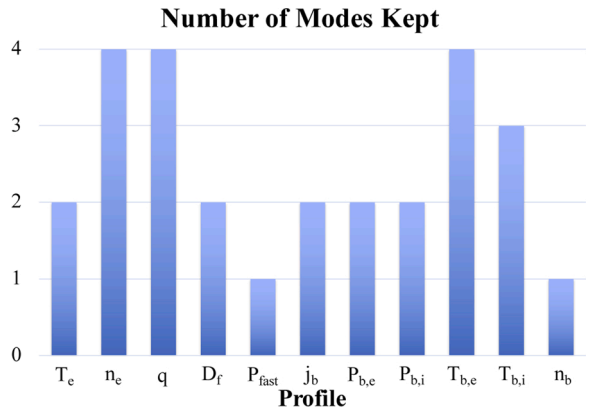


Fig. 3. Number of basis functions retained after PCA for each profile.

test will better reflect future uses. Data for each input and output was extracted from the TRANSP runs and saved in either a training, testing, or validation batch. Tables 1 and 2 provide a detailed list of inputs and outputs, respectively. The inputs shown in Table 1 describe the equilibrium, the kinetic and magnetic states of the plasma, and the neutral beam settings, all of which are expected to have a direct impact on neutral beam injection according to the physics of the plasma. The selection of the set of inputs is finalized when the neural network is able to learn the function connecting these inputs with the desired outputs in Table 2, proving that the desired outputs are in fact functions of these inputs. Fig. 2 shows the range of data in the training set for each input. This is a lower-dimensional representation of the domain for which the neural network predictions are valid.

2.1. Principal component analysis of profile data

To maximize the number of potential applications, ranging from off-line simulation to real-time optimization, the goal of this work was to generate a model with as fast a computation time as possible without significantly sacrificing prediction accuracy. The need of reducing complexity, as dictated by this goal, motivated the decision of not using a convolutional neural network model, which is well-suited to two dimensional spatially-varying data and often used in image recognition applications. A simpler multilayer perceptron network was used instead. This is possible because the profile variables are one dimensional, only varying in the radial direction, and can be reduced to a set of scalar values using principal component analysis (PCA). This method projects each radially varying quantity onto a set of basis functions, and determines which basis functions explain the majority of the variance. By using a reduced number of basis functions as inputs and outputs of the network, a similar level of accuracy is achieved while significantly reducing the number of nodes in the input and output layers of the model. Like the neural network itself, the PCA is trained on the training dataset to determine the basis functions, and does not perform well for profiles that are significantly different from any profile in the training dataset. In this model, modes explaining at least 0.5% of the variance in the data were kept, and all other modes were neglected. Fig. 3 shows the number of modes kept to represent each one of the spatial profiles. Over 98.5% of the data variance was kept for each profile, which ensured that the dominant features of the profiles were retained while the neglected modes mostly contained "noise." By reducing the amount of "noise" in the data, the principal component analysis smoothes out the output profiles and helps to avoid overfitting.

2.2. Beam slowing down time effects

The effects of neutral beam injection on the plasma are dependent on the time history of the discharge due to the fact that it takes a

measurable amount of time for the fast ions injected by the beam to transfer their energy to other particles. Each time a fast ion collides with a thermal ion, some of the energy carried by the fast ion is transferred to the thermal ion, slowing down the fast ion. A commonly employed method to model time history is the recurrent neural network. However, this approach was discarded for the same reason as the convolutional neural network was discarded: the architecture is more complicated, making the calculation time significantly longer. Instead, using the same process as in [12], the inputs to the model are augmented with a set of causal low-pass filtered versions of the individual beam powers. The beam powers are filtered with time constants $\tau_{LP} = 0.02$ s, 0.05 s, and 0.1 s to account for a range of different potential slowing down times. The first-order filter

$$\dot{x} = \frac{P_{inj} - x}{\tau_{LP}} \tag{2}$$

is used where x is the filtered output.

2.3. Standardization

In neural networks, standardization of input and output data is required to ensure that proper weight is given to each input, regardless of units. Without any scaling of the inputs, more weight would be given to inputs with a higher absolute value. In addition, lack of scaling of the outputs could cause one output to overwhelm the others in the calculation of the loss function. In this model, once the principal component analysis has been applied to each profile and the beam slowing down effects have been accounted for, each input feature is independently standardized to a mean of 0 and a variance of 1. The predicted output from the model is returned to its true magnitude through the inverse of the standardization process. Then the predicted scalar values for the principle component analysis are mapped back to one dimension to produce recognizable spatial profile predictions.

3. Determination of model architecture

The neural networks in this work use a fully connected structure, meaning that each node feeds into every node in the next layer. As stated in Section 1, when building a neural network, the weights of each connection between nodes are initially randomized, and adjusted through the training process. In order to account for the uncertainty caused by the initial randomization, five separate neural networks were trained in parallel with different initial weights as illustrated in Fig. 4. When calling a prediction, each of the five models makes its own prediction, and the average, standard deviation, minimum, and maximum of the five predictions are returned. Fig. 5 shows the correlations, described by the R^2 values of a linear regression, between the average neural-network prediction and the NUBEAM prediction for different numbers of parallel neural-network models used. Five parallel models are chosen in this work as a tradeoff between prediction accuracy and computational time, which both grow as the number of parallel models is increased. In applications where a faster calculation time is more important than higher prediction accuracy, the users have the flexibility to choose how many of the parallel models they run, or to use parallel computation, to meet their calculation time requirements.

Currently, there is no concrete way to determine the optimal values of manually-defined hyperparameters such as the number of hidden layers, the number of nodes per hidden layer, and the length of the training process in a neural network, although this is an area of research (e.g., genetic-algorithm optimization [17], Bayesian optimization [18]). Instead, standard practice is to conduct a grid search, a brute force testing of different combinations of the hyperparameters to see which values give the best results. Fig. 6 shows the results of testing different numbers of hidden layers and numbers of nodes per hidden layer. Plots (a) and (c) show the correlations between the neural network

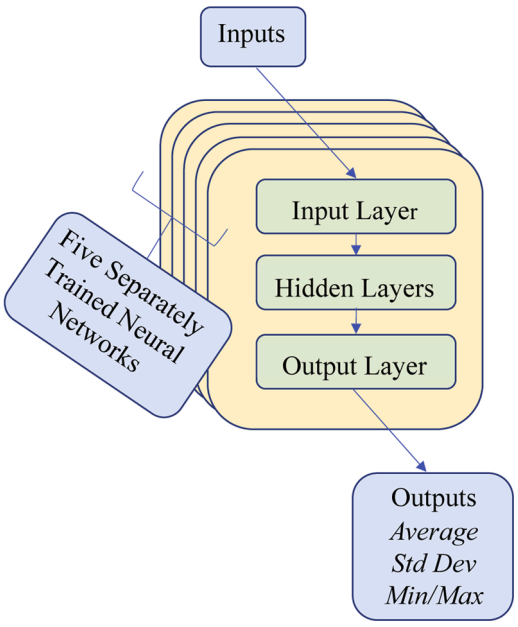


Fig. 4. Structure of the neural network model.

predictions and the actual NUBEAM data for runs in the validation data set, and plots (b) and (d) show how long each prediction took to be calculated. Plots (a) and (b) show results as a function of the number of hidden layers and number of neurons per hidden layer in the model, and plots (c) and (d) show results as a function of the number of weights the network has to learn. Note that these predictions were called in the modeling framework OMFIT [19] on the Iris cluster at General Atomics using Python, and are intended to show the difference in calculation times between different architectures, not the absolute value of calculation time. The grid search shows very little difference in accuracy between 2 and 3 hidden layers when up to 125 nodes per layer are used, with the 1 layer models showing significantly lower R^2 values. When more nodes are used, both the 2 and 3 hidden layer models show a decrease in R^2 as the network complexity increases, indicating overfitting. The overfitting due to an increased network complexity can also be seen by the decrease in accuracy when the number of learned parameters exceeds ~25,000. The highest R^2 value is achieved using 2 layers of 125 nodes per layer. However, the calculation time begins to increase for models with 100 nodes per layer. In order to minimize the calculation time as much as possible while still predicting with good accuracy, an architecture with 2 hidden layers of 75 nodes each was

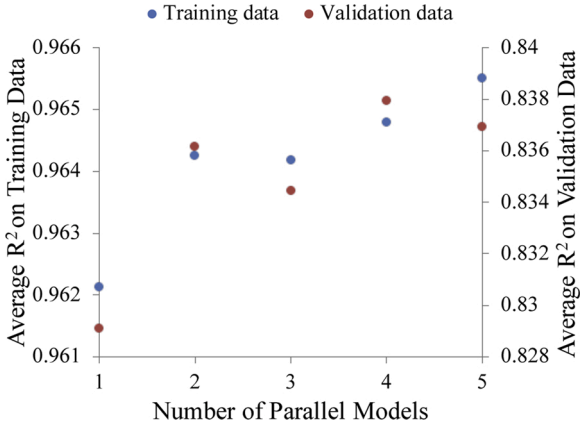
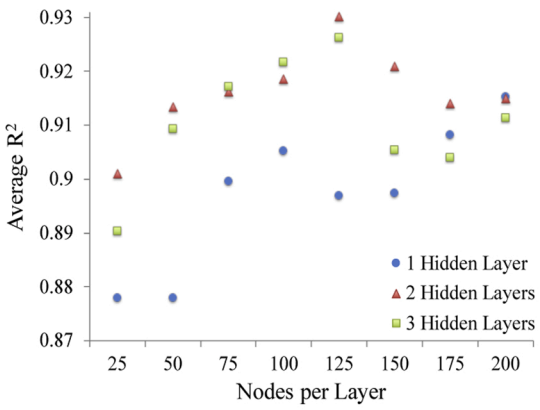
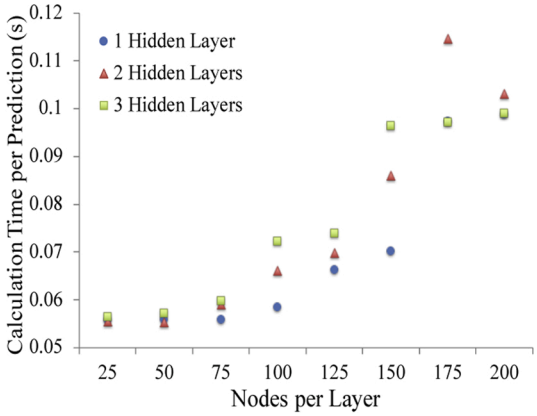


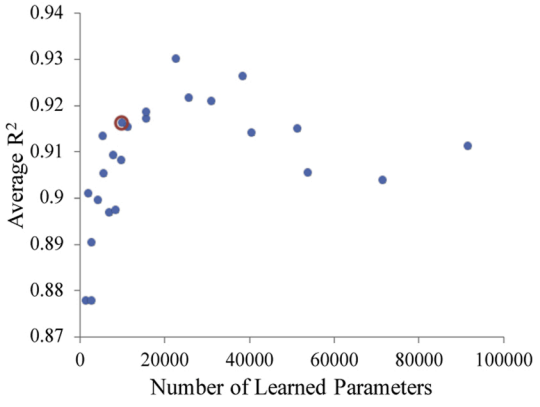
Fig. 5. Average R^2 for training and validation data as a function of the number of parallel models used to find average prediction value.



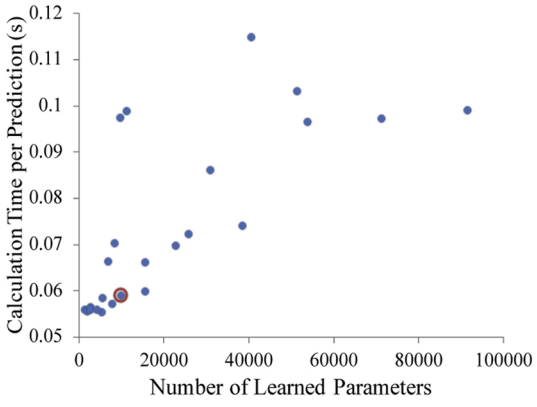
(a) Average R^2 as a function of architecture.



(b) Computation time as a function of architecture.



(c) Average R^2 as a function of parameters.



(d) Computation time as a function of parameters.

Fig. 6. (a) Computation accuracies averaged across all outputs as a function of model architecture (number of layers and nodes per layer), (b) prediction times as a function of model architecture (number of layers and nodes per layer), (c) computation accuracies averaged across all outputs as a function of the number of learned parameters, and (d) prediction times as a function of the number of learned parameters. In plots (c) and (d), the circled values represent 2 hidden layers with 75 nodes per layer, which is the architecture chosen for the final network.

adopted as an acceptable compromise solution.

Another grid search was conducted using this architecture to determine suitable values for the model training parameters: number of epochs and batch size. The network is trained for a set number of epochs, or number of times the entire training data set is passed forward and backward through the network. In Fig. 7, the blue dashed line shows the R^2 value for predictions in the validation dataset. The red solid line shows the difference between the R^2 values for predictions in the training dataset and predictions in the validation dataset. Predictions on the training data will always be more accurate than on data the network has never seen before; however, too great a difference can indicate underfitting or overfitting. Underfitting is when the network has not fully learned the correlations in the data; overfitting is when the network

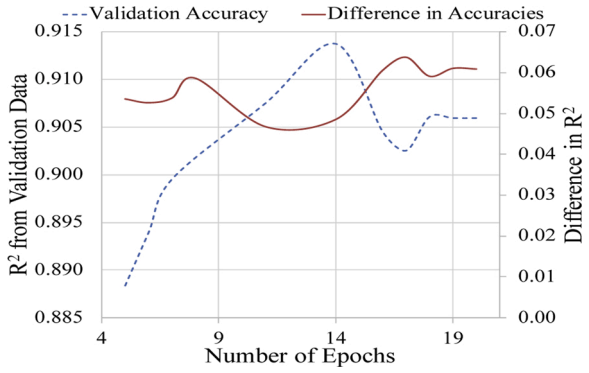


Fig. 7. Prediction accuracy on the validation dataset and difference in accuracy between training and validation data vs. number of epochs. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

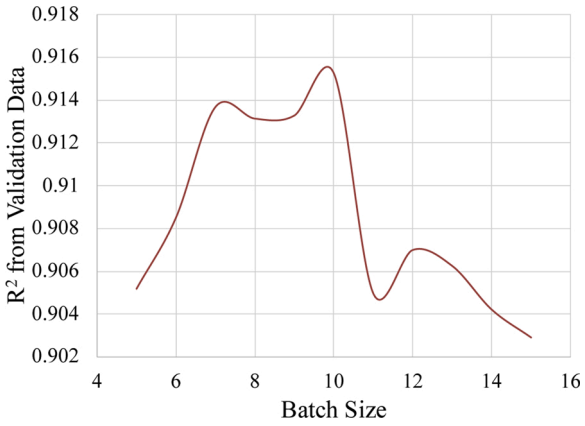


Fig. 8. Prediction accuracy on the validation dataset vs. batch size.

Table 3
Hyperparameters used in the final neural network model.

Number of hidden layers	2
Nodes per hidden layer	75
Number of epochs	14
Batch size	10

Table 4
Keras functions used in the final neural network model.

Solver	'sgd'
Hidden layer activation function	'relu'
Output layer activation function	'linear'
Loss	'mse'
Metrics	'accuracy'

has learned the specific training data too well instead of learning the underlying correlations. In order to maximize the accuracy on the validation dataset and minimize the difference in accuracies, the number of epochs should be chosen where the R^2 for the validation data peaks and the difference in R^2 is low; in the case of this model, that happens when the model is trained for 14 epochs. Training efficacy is also highly dependent on the batch size, or the number of data points seen by the network between each update of the weights. Based on the results shown in Fig. 8, the prediction accuracy peaks for a batch size of 10, so this value was chosen for the final model. The values of all hyperparameters determined by the grid search are shown in Table 3. Table 4 lists the values of all other inputs to the Keras [20] network used in this model.

4. Model evaluation

Results from the model as described in Section 3 are shown in this section. Fig. 9 shows predictions of (a) current drive, (b) fast ion pressure, (c) beam heating to electrons, and (d) neutron rate plotted vs. time. These results are from TRANSP run 175282T01. This run is from the testing dataset, so the neural network did not see it during training. In the plots, the red dashed line shows the NUBEAM data, the dark blue solid line shows the average prediction across all five trained neural networks, and the blue shaded area shows one standard deviation above and below the average of the five predictions. The beam current drive, fast ion pressure, and beam heating to electrons profiles are plotted at the spatial location $x = 0.211$, where x denotes the normalized toroidal

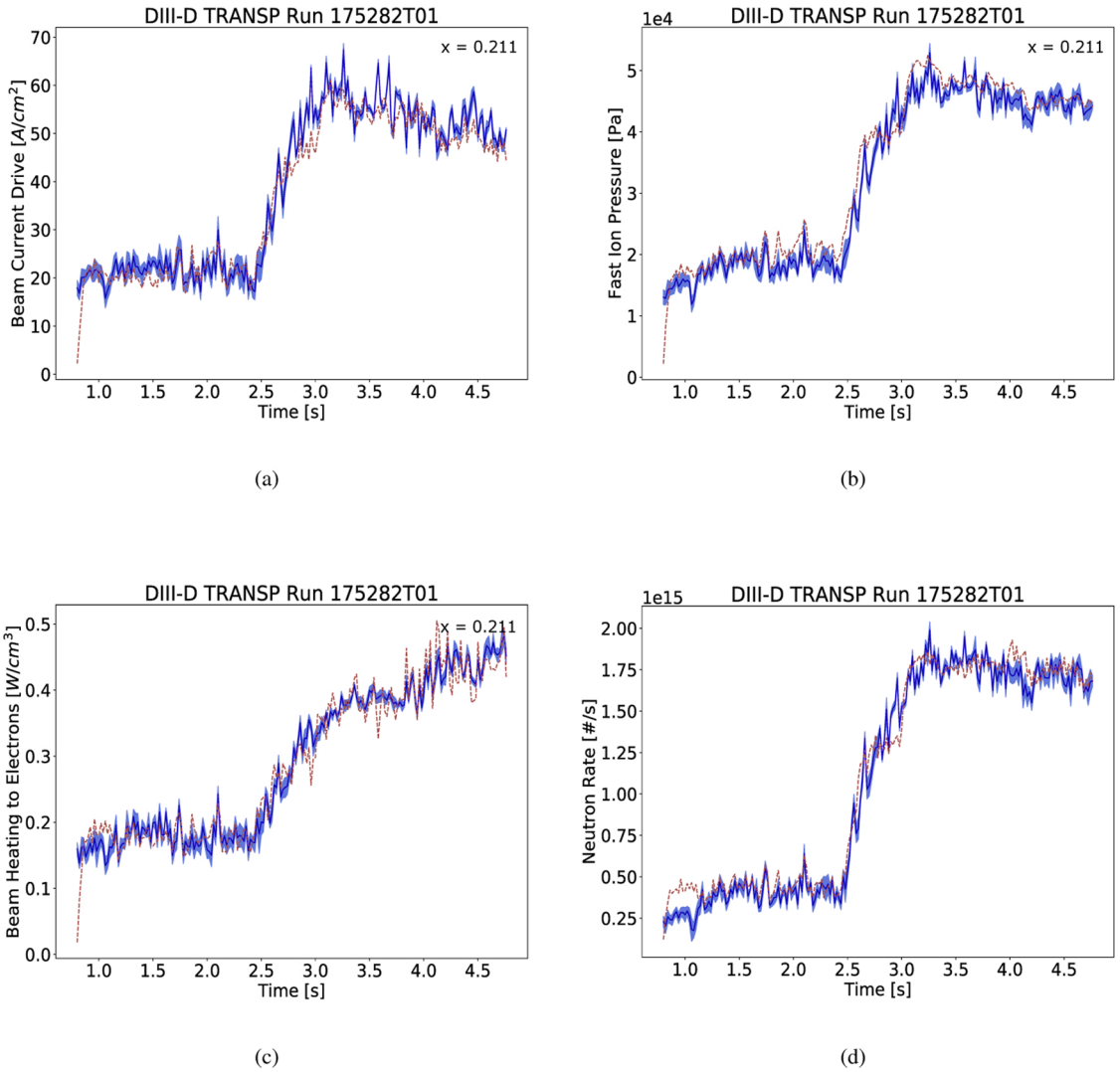


Fig. 9. Time traces of NUBEAM and NubeamNet predictions for (a) current drive($x = 0.211$), (b) fast ion pressure ($x = 0.211$), (c) heating to electrons ($x = 0.211$), and (d) neutron rate for TRANSP run 175282T01. Red dashed line is NUBEAM data, blue solid line is NubeamNet average prediction, blue shaded area is one standard deviation of NubeamNet prediction. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

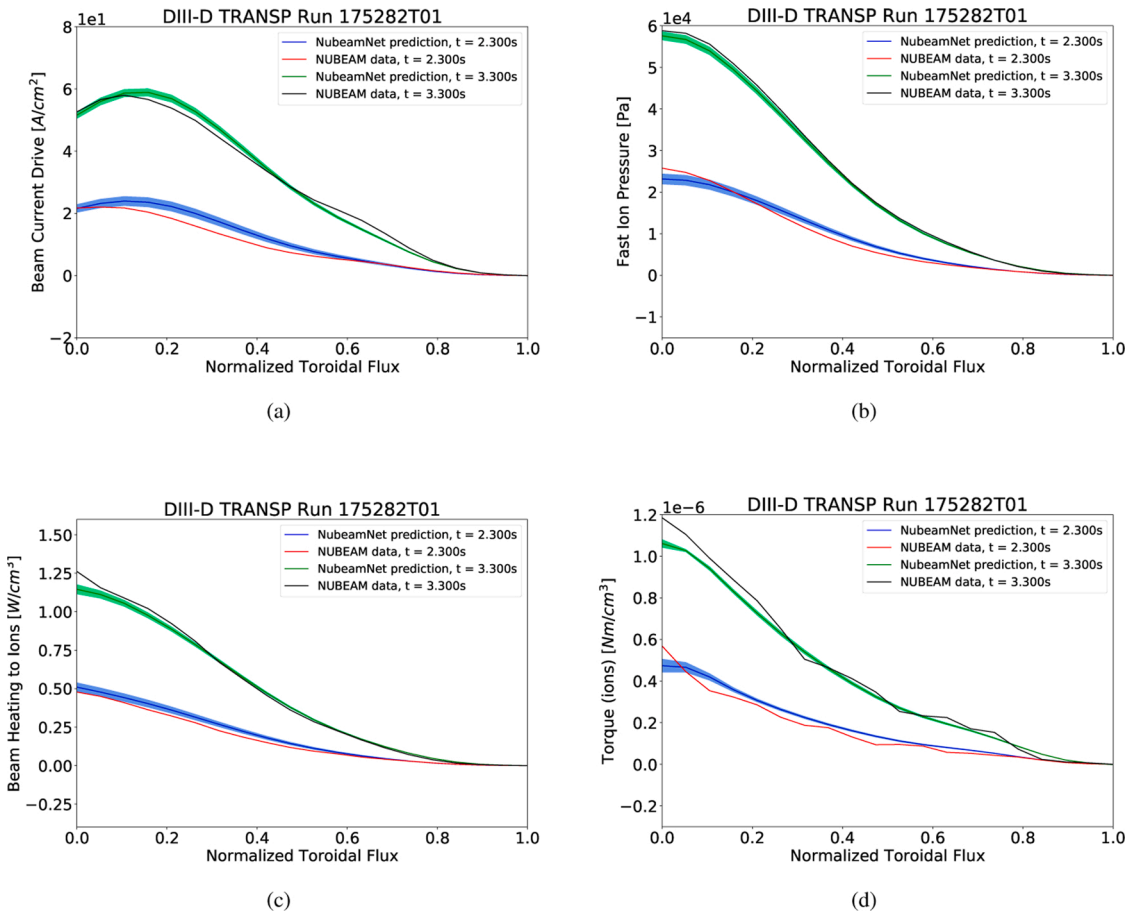


Fig. 10. Predictions of profiles from NUBEAM and NubeamNet plotted as functions of the normalized toroidal flux for (a) current drive, (b) fast ion pressure, (c) heating to ions, and (d) torque to ions for TRANSP run 175282T01 at times $t = 2.3$ s and $t = 3.3$ s. The solid blue and green lines represent the average predictions across all five networks, and the shaded areas show one standard deviation above and below the average. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

flux. The time traces show good agreement across the shot, and quick responses to changes in the beam powers. Fig. 10 shows prediction of (a) current drive, (b) fast ion pressure, (c) beam heating to ions, and (d) torque to ions plotted vs. normalized toroidal flux for the same run at time $t = 2.3$ and 3.3 s. The NUBEAM data for torque is spatially noisy, which stems from the Monte Carlo nature of the code. The noise can be limited by increasing the fidelity of the simulation, but it will never fully disappear. Good agreement is shown in the spatial profiles, and the shapes of the profiles are accurately reproduced despite the reduced number of modes used in the principle component analysis. Note that, for the purposes of training, NUBEAM data is assumed to be correct. The neural network is trained only on NUBEAM data, and no attempt is made to quantify the accuracy of the neural network outputs to experimental data. As shown in Fig. 10, NubeamNet produces a smoother prediction of the torque profiles than NUBEAM. Physically, this profile should be smooth, so it is possible that the NubeamNet prediction is actually closer to experimental data than the NUBEAM data; however, this work does not attempt to quantify whether the NubeamNet prediction is truly more accurate or just smoother. Fig. 11 shows the NubeamNet prediction plotted against the NUBEAM data for each data point in the testing dataset for (a) heating to ions, (b) heating to electrons, (c) fast ion pressure, and (d) neutron rate. All plots show the highest concentration of data points along the line where the NubeamNet prediction is equal to the NUBEAM data. In the plots of profiles, a distinct branch can be seen below the line along which the NubeamNet prediction is equal to the NUBEAM data. This branch is caused by a few specific shots for which the network consistently predicts heating, fast ion pressure, and torque outputs below the correct value. Future work will include adding more shots with similar plasma scenarios to the training dataset to improve predictions on these shots.

Table 5 shows the R^2 value of the correlation for each output for data in the training dataset and testing datasets. The correlations are slightly higher for the training data, but the difference is generally very small. For a few of the outputs (orbit loss, power to CX, and CX SCE) the difference between the R^2 values on the training and testing datasets is more significant. This could be due, for example, to a lower variance in the data in the much smaller testing set, as R^2 is inherently a measure of explained variance. Although these differences are not ideal, the relatively low differences between training and testing set results for the other outputs indicates that they are most likely not caused by significant overfitting. In addition, the R^2 values for torque to ions are lower than for other outputs for both the training and testing datasets. This is due to the fact that the NUBEAM torque data is noisy, meaning that the maximum achievable R^2 value is lower. Overall, these results indicate that the model will perform well on data it has not seen before, provided the new data is still within the range of the training data.

The final model takes an average of 1.3 ms to predict a single time step using all five parallel models, and 0.42 ms using just one model, when called using Cython [21]. This calculation time is roughly 20 times faster than the time reported for the reduced analytical model shown in [16], and orders of magnitude faster than NUBEAM. It is also likely that the calculation time will be further reduced when the network is called using C on the real-time DIII-D Plasma Control System (PCS) computer.

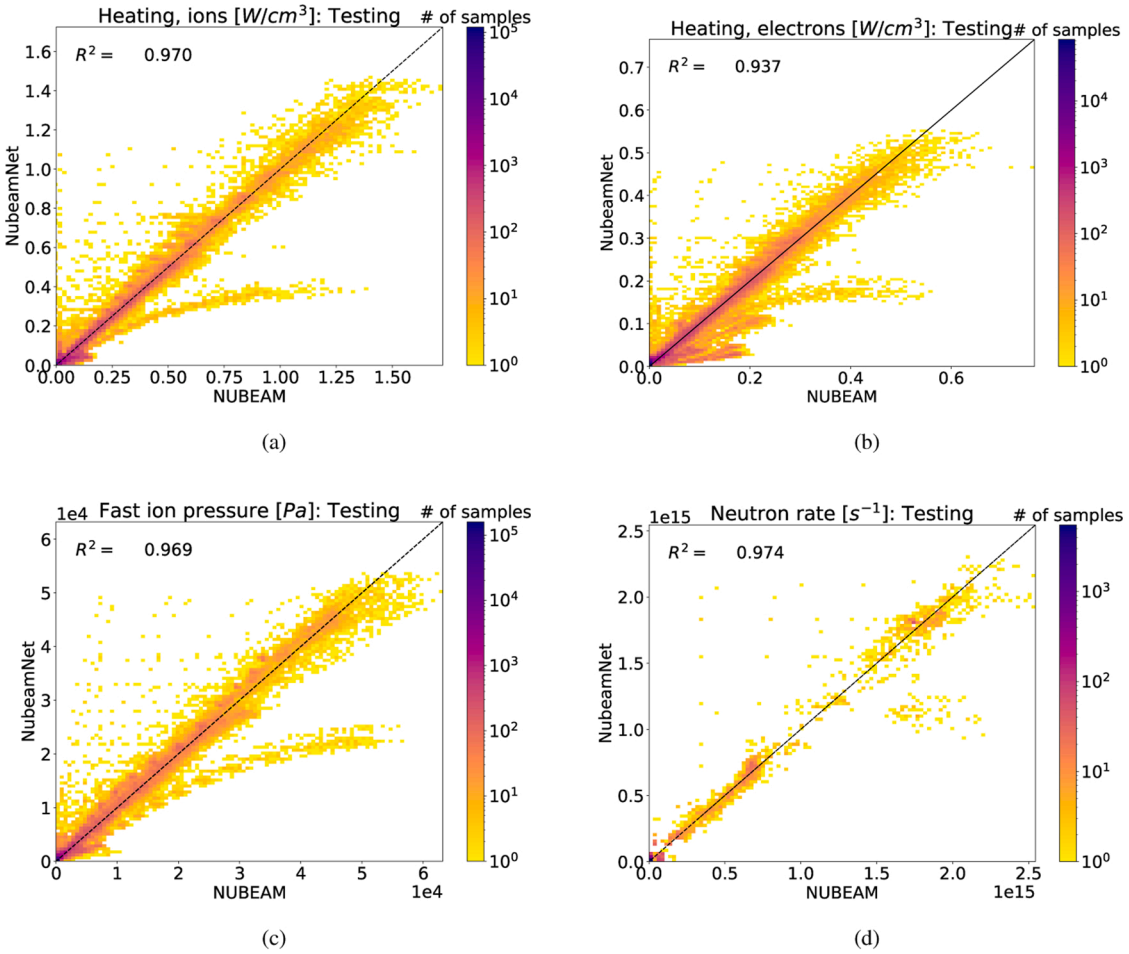


Fig. 11. Log-scale plots of regression results comparing NUBEAM and NubeamNet predictions for shots in the testing dataset for (a) heating to ions, (b) heating to electrons, (c) fast ion pressure, and (d) neutron rate. Data points in plots (a), (b), and (c) represent values at discrete spatial locations within the profiles.

Table 5
Correlations (R^2) between NUBEAM and NubeamNet predictions for shots in training and testing datasets.

	R^2 values: training data	R^2 values: testing data
Neutron rate	0.984	0.974
Shine through	0.983	0.991
Orbit loss	0.940	0.888
Power to CX	0.985	0.847
CX SCE Power	0.986	0.857
Fast ion pressure	0.987	0.969
Beam driven current	0.982	0.961
Heating of electrons	0.971	0.937
Heating of ions	0.987	0.970
Torque to electrons	0.954	0.945
Torque to ions	0.841	0.874
Beam ion density	0.986	0.965

5. Conclusions

A neural network model for calculating the effects of neutral beam injection in DIII-D has been developed. The model was trained on simulation data from NUBEAM derived from TRANSP runs for shots from the DIII-D 2018 campaign. The speed of predictions makes it useful for applications demanding fast simulations such as optimal scenario planning between discharges. The calculation time also suggests that the neural-network model may be useful for real-time control applications such as feedback control via real-time optimization, state estimation, and forecasting. The predictions are shown to have a high level of accuracy when compared to NUBEAM results.

Future work will include integrating the neural-network model into the DIII-D PCS for use in real-time model-based control applications. To achieve this goal, the model will be integrated first into the Control Oriented Transport SIMulator (COTSIM) code developed by the Lehigh University Plasma Control Group. COTSIM, which is coded in Matlab® /Simulink®, is a modular 1D transport code designed to run fast enough for control applications such as iterative control design, model-based scenario optimization, and real-time control/estimation/forecasting. COTSIM currently uses empirical scaling laws to model the neutral beam heating, current drive, and torque. As an alternative to these scaling laws, which are scenario specific, the proposed neural-network model has the potential of both increasing the prediction accuracy of COTSIM and broadening its applicability range without the need of retuning the model.

Authors' contributions

Shira Morosohk: Methodology, Software, Investigation, Writing – Original Draft

Dan Boyer: Conceptualization, Methodology, Software

Eugenio Schuster: Conceptualization, Writing – Review and Editing, Supervision

Conflict of interest

The authors declare no conflict of interest.

Declaration of Competing Interest

The authors report no declarations of interest.

Acknowledgment

This work has been supported by the U.S. Department of Energy, Office of Science, Office of Fusion Energy Sciences, under Award DE-SC0010661, and by the National Science Foundation Graduate Research Fellowship Program under Grant No. 1842163.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at https://doi.org/10.1016/j.fusengdes.2020.112125.

References

[1] R. Hawryluk, Physics of Plasmas Close to Thermonuclear Conditions, Pergamon, 1981.
[2] J. Breslau, M. Gorelenkova, F. Poli, J. Sachdev, X. Yuan, Transp, [Computer Software], 2018, https://doi.org/10.11578/dc.20180627.4.
[3] Y. Ou, T.C. Luce, E. Schuster, J.R. Ferron, M.L. Walker, C. Xu, D.A. Humphreys, Towards model-based current profile control at DIII-D, Fusion Eng. Des. 82 (2007) 1153–1160.
[4] G.E. Hinton, How neural networks learn from experience, Sci. Am. 267 (3) (1992) 144–151. https://www.jstor.org/stable/10.2307/24939221.
[5] K. Hornik, Approximation capabilities of multilayer feedforward neural networks, Neural Netw. 4 (1991) 251–257, https://doi.org/10.1016/0893-6080(91)90009-T.
[6] M.A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.
[7] O. Meneghini, et al., Self-consistent core-pedestal transport simulations with neural network accelerated models, Nucl. Fusion 57 (086034) (2017).
[8] J. Citrin, et al., Real-time capable first principle based modeling of tokamak turbulent transport, Nucl. Fusion 55 (092001) (2015).
[9] F. Felici, et al., Real-time-capable prediction of temperature and density profiles in a tokamak using RAPTOR and a first-principle-based transport model, Nucl. Fusion 58 (096006) (2018).
[10] R. Goldston, et al., New techniques for calculating heat and particle source rates due to neutral beam injection in axisymmetric tokamaks, J. Comput. Phys. 43 (1981) 61–78.
[11] A. Pankin, et al., The tokamak Monte Carlo fast ion module NUBEAM in the National Transport Code Collaboration library, Comput. Phys. Commun. 159 (2004) 157–184.
[12] M. Boyer, et al., Real-time capable modeling of neutral beam injection on NSTX-U using neural networks, Nucl. Fusion 59 (056008) (2019).
[13] A. Pajares, et al., Integrated current profile, normalized beta and NTM control in DIII-D, Fusion Eng. Des. 164 (2019) 559–562.
[14] A. Pajares, et al., Nonlinear Robust Safety Factor Profile Control in Tokamaks Via Feedback Linearization and Nonlinear Damping Techniques, 2018.
[15] M. Boyer, et al., Feedback control of stored energy and rotation with variables beam energy and perveance on DIII-D, Nucl. Fusion 59 (2019).
[16] M. Weiland, et al., RABBIT: real-time simulation of the NBI fast-ion distribution, Nucl. Fusion 58 (082032) (2019).
[17] G.-C.V.P.G. Bernardos, Optimizing feedforward artificial neural network architecture, Eng. Appl. Artif. Intell. 20 (2007) 365–382.
[18] A. Klein, S. Falkner, S. Bartels, P. Hennig, F. Hutter, Fast Bayesian optimization of machine learning hyperparameters on large datasets, in: Vol. 54 of Proceedings of Machine Learning Research, PMLR, Fort Lauderdale, FL, USA, 2017, pp. 528–536, in: http://proceedings.mlr.press/v54/klein17a.html.
[19] O. Meneghini, et al., Integrated modeling applications for tokamak experiments with OMFIT, Nucl. Fusion 55 (8) (2015) 083008.
[20] F. Chollet, et al., Keras, 2015. https://github.com/fchollet/keras.
[21] S. Behnel, R. Bradshaw, L. Dalcin, M. Florisson, V. Makarov, D.S. Seljebotn, C-Extensions for Python, https://cython.org/.