Neural network tokamak equilibria with incompressible flows

D. A. Kaltsas[1, 2, a)] and G. N. Throumoulopoulos[1, b)]

[1)]*Department of Physics, University of Ioannina,*
GR 451 10 Ioannina, Greece
[2)]*Department of Physics, International Hellenic University,*
GR 654 04 Kavala, Greece

(Dated: 14 December 2021)

We present several numerical solutions to a generalized Grad-Shafranov equation (GGSE), which governs axisymmetric plasma equilibria with incompressible flows of arbitrary direction, using fully connected, feed-forward, deep neural networks, also known as multi-layer perceptrons. Such artificial neural networks (ANNs) are trained to approximate tokamak-relevant equilibria upon minimizing the GGSE mean squared residual in the plasma volume and the poloidal flux function on the plasma boundary. Solutions for the Solovev and the general linearizing ansatz for the free functions involved in the GGSE are obtained and benchmarked against known analytic solutions. We also construct a non-linear equilibrium incorporating characteristics relevant to the H-mode confinement. In our numerical experiments it was observed that changing the radial distribution of the training points has surprisingly small effect on the accuracy of the trained solution. In particular it is shown that localizing the training points at the plasma edge results in ANN solutions that describe quite accurately the entire magnetic configuration, thus demonstrating the interpolation capabilities of the ANNs.

I. INTRODUCTION

A. The Generalized Grad-Shafranov Equation

Equilibrium and stability of fusion plasmas are commonly studied in the context of ideal Magnetohydrodynamics (MHD); in the case of tokamaks, which are toroidal and nearly axisymmetric magnetic confinement devices, the description is simplified geometrically by the assumption of axial symmetry. Equilibrium studies in the context of axisymmetric MHD rely on the solution of the so-called Grad-Shafranov equation (GSE)[1,2], which is usually solved for static plasmas or for purely toroidal plasma flows. For arbitrary compressible flows, a nonlinear coupling between the GSE and the Bernoulli equation emerges, hence elaborate equilibrium solvers (see, e.g., Ref. [3]) are required to determine equilibrium solutions. Macroscopic sheared flows are important for plasma confinement since they are associated with the suppression of certain instabilities and the formation of transport barriers that reduce the radial transport leading to improved confinement modes (High confinement mode – H-mode) in tokamaks[4]. Therefore, a flexible modelling of sheared flows of arbitrary direction, particularly in connection with the H-mode phenomenology is desirable. Such a flexible and simplified description is provided by introducing the assumption of flow incompressibility, that decouples the GSE and the Bernoulli equation and ensures that the former stays in the elliptic regime. This implies that the incompressibility assumption is limited to low Mach number flows.

It is known that stationary MHD states of axisymmetric plasmas with incompressible flows satisfy the following generalized Grad-Shafranov equation (GGSE)[5,6]

$$\Delta^* u + \frac{1}{2}\frac{d}{du}\left[\frac{X^2}{1-M_p^2}\right] + r^2\frac{dP_s}{du} + $$
$$+ \frac{r^4}{2}\frac{d}{du}\left[\rho\left(\frac{d\Phi}{du}\right)^2\right] = 0\,, \qquad (1)$$

where $\Delta^* = r^2\nabla\cdot(\nabla/r^2)$ and (r,ϕ,z) are cylindrical coordinates with z corresponding to the axis of symmetry. The function $u = u(r,z)$ that labels the magnetic surfaces is associated with the ordinary magnetic flux function ψ via the following integral transformation[7]

$$u(\psi) = \int_0^\psi \left[1 - M_p^2(g)\right]^{1/2} dg\,. \qquad (2)$$

Here, $M_p(u)$ is the Mach function of the poloidal fluid velocity with respect to the poloidal Alfvén velocity. In Eq. (1), $X(u)$ relates to the toroidal magnetic field, $B_\phi = I/r$, through $I = (X - r^2\sqrt{\varrho}M_p\Phi')/(1-M_p^2)$; $\Phi(u)$ is the electrostatic potential and $\rho(u)$ the plasma density; for vanishing flow the surface function $P_s(u)$ coincides with the pressure and Eq. (1) reduces to the usual (static) GS equation.

For fixed boundary equilibria, Eq. (1) is accompanied by a Dirichlet boundary condition of the form $u|_{\partial\mathcal{D}} = u_b$, where u_b is a fixed value and $\partial\mathcal{D}$ is the boundary of the domain $\mathcal{D} \subset \mathbb{R}^2$. In present-day tokamaks the plasma boundary is of a characteristic D shape which is elongated in the vertical direction and usually has a lower X-point associated with the presence of a plasma divertor. A flexible and accurate approximation of the computational domain and its boundary, which might be up-down symmetric or asymmetric, is required for constructing accurate tokamak equilibria. These equilibrium states are important for understanding and improving plasma confinement since they are used not only for studying

force-balance but also stability, and transport phenomena. In two previous papers we derived analytic solutions in analytically prescribed domains for the Solovev[8] and the general linearizing ansatz[9,10] deploying an effective method for boundary shaping[11]. However, these solutions are limited in the linear regime and finding solutions for nonlinear choices of the free functions in general requires the deployment of numerical methods, which are usually based on finite difference or finite element discretizations of the computational domain. For solutions obtained by conventional numerical methods, a sophisticated discretization is often required so as the boundary to be approximated with sufficient accuracy. In addition, these are not continuous, closed form solutions, so their evaluation in between the discrete nodes requires interpolation and the analysis is grid-dependent.

Motivated by the growing interest in neural network and machine learning techniques for solving differential equations, in this work we apply these ideas in the case of the GGSE, demonstrating the capability of neural network solutions to describe efficiently tokamak equilibria with flows of arbitrary direction, solving Eq. (1) for fixed boundaries. The method is accurate and grid-independent since no sort of discretization for the domain $\mathcal{D}$ and derivative approximations are needed as in classical numerical methods. This is accomplished upon using automatic differentiation (AD)[12,13] to compute the partial derivatives of the neural network solution with respect to the input coordinates. For the training process we use randomly sampled points, and impose the boundary conditions on training points that are generated by parametric equations, thus giving us a great flexibility in shaping the boundary as desired. The method works efficiently for various boundary shapes and also for nonlinear GGSEs providing closed form ANN solutions, which are continuous and differentiable. Therefore, it is a potential candidate for treating some of the pathologies of the standard analytic and numerical methods.

Neural Network Grad-Shafranov equilibria have already been constructed in the past, e.g. in Ref. [14] and in Ref. [15] which was concerned with the reconstruction of magnetic equilibria using measured magnetic signals. Neural network representations of the magnetic flux were also employed in Ref. [16] for rapid recovery of the plasma topology. In Ref. [14] the authors employed a procedure quite similar to the method described in Ref. [13], which is essentially the approach utilized also in the present paper. In this work though, we solve the generalized GS equation (1) rather than the static GSE and in addition, we use Deep Neural Networks (DNNs) in contrast to Ref. [14] where MLPs with only one hidden layer were employed. Also, our approach allows for analytically defined domains while is completely mesh-free, since the training points are randomly sampled, and aims to highlight the interpolation capabilities of the ANN solutions. These remarkable interpolation properties are demonstrated upon considering both radially uniform distributions of points, uniform in the sense that the point density is approximately constant throughout the computational domain, and point distributions localized near the plasma edge. Remarkably, the solutions obtained by these two settings have very small relative errors, with maximum values in the central region of the domain where in the second setting there are no training points.

The rest of the paper is organized as follows: in Sec. II we present some basic notions and methods regarding the ANN solutions to Partial Differential Equations (PDEs), then in Sec. III we describe the methodology we implemented for this particular problem, in Sec. IV the main results obtained by our numerical experiments are presented, and in Sec. V we summarize the conclusions and discuss possible future extensions.

II. ARTIFICIAL NEURAL NETWORKS AND PDEs

The idea of using ANNs and in particular MLPs to represent solutions to PDEs is documented in the seminal work of Lagaris et. al. in Ref. [17] and is based on the universal approximation theorem[18] which guarantees that MLPs are universal function approximators. Interestingly, this approach was exploited even earlier for the static Grad-Shafranov equation in Ref. [14] utilizing MLPs with only one hidden layer. Over the last few years this subject field has attracted an explosively growing interest due to recent developments involving, e.g., the so-called Physics Informed Neural Networks (PINNs)[13]. The neural network solutions are still approximate, however, unlike conventional numerical solutions, they are continuous and their output values can be recovered everywhere inside the computational domain, once the weights and biases of the neural network are determined. Due to this growing interest in using neural networks to approximate solutions to boundary and initial value problems, several software packages have been developed in the last couple of years, for example we are aware of DeepXDE, NeuroDiffEq, PyDEns and Nangs packages[19–22]. The results of the present study though, are obtained by an implementation built from scratch so that we could easily adapt the code to the needs of our specific problem. Although more information on these methods can be found, for example, in Refs. [13], [17], and [23], for reasons of completeness we describe here some of the basic principles for solving differential equations using neural networks.

The central idea is to turn the PDE and the accompanying boundary conditions, into an optimization problem where we aim to minimize the residual of the differential equation and the residual of the boundary conditions. A boundary value problem, like the fixed-boundary MHD equilibrium with incompressible flows, described by Eq. (1) and Dirichlet boundary conditions for u, can be written in the form

$$\mathcal{R} := \mathcal{L}u - S(\mathbf{x}, u) = 0\,, \quad in\ \mathcal{D} \subset \mathbb{R}^n$$
$$u(\mathbf{x}) - u_b = 0\,, \quad on\ \partial\mathcal{D}\,. \tag{3}$$

A neural network solution $u_n = u_n(\mathbf{x}; \mathbf{p})$, which is not an exact solution, produces non vanishing residual errors when substituted in (3). Here, $\mathbf{p}$ represents a vector composed of the neural network parameters. These errors in the continuous setting are estimated by

$$L_i(\mathbf{p}) = \frac{1}{V_\mathcal{D}} \int_\mathcal{D} d^n x \left\| \mathcal{L}u_n(\mathbf{x}; \mathbf{p}) - S(\mathbf{x}, u_n(\mathbf{x}; \mathbf{p})) \right\|^2, \quad (4)$$

$$L_b(\mathbf{p}) = \frac{1}{S_{\partial\mathcal{D}}} \int_{\partial\mathcal{D}} d^{n-1} x \left\| u_n(\mathbf{x}; \mathbf{p}) - u_b \right\|^2, \quad (5)$$

$$(6)$$

Then to perform the optimization, the two loss functions are approximated using Monte-Carlo integration using a set of randomly sampled points. These points serve as the training data that are fed to the neural network whose parameters, i.e., the weights and the biases, are adapted so as the two loss functions are minimized. Note that the errors in the discrete problem are reduced to the Mean Squared Errors (MSEs). The minimization is effected by some gradient-based optimization method, e.g., the Gradient Descent method leveraging the backpropagation algorithm to compute the derivatives of the loss functions with respect to the network parameters. Usually a weighted sum of the two losses is minimized, i.e.

$$L(\mathbf{p}) = \lambda_i L_i(\mathbf{p}) + \lambda_b L_b(\mathbf{p}). \quad (7)$$

Having formed the loss function (7), the optimization procedure can be expressed mathematically as follows:

$$\min_{\mathbf{p}} (L(\mathbf{p})). \quad (8)$$

1. Neural Network Architecture

We have already mentioned that an MLP is a feed-forward, fully connected neural network with multiple hidden layers. In such ANNs the information moves only in one direction from the input to the output layer. Each neuron is connected with the neurons of the previous layer by linear functions, containing weights and amplitudes, called biases. These are the learnable model parameters, which are adapted (learned) during the training process. The output of each neuron is filtered by a nonlinear function, the so-called activation function f, before is fed to the next layer of neurons. Overall, an MLP can be written as a composition of alternating linear and non-linear transformations

$$u_n(\mathbf{x}; \mathbf{p}) = \mathbf{W}^{(\ell)} \cdot f^{(\ell)}(\mathbf{W}^{(\ell-1)} \cdot f^{(\ell-1)}(\cdots$$
$$f^{(1)}(\mathbf{W}^{(0)} \cdot \mathbf{x} + \mathbf{b}^{(0)}) \cdots) + \mathbf{b}^{(\ell-1)}) + \mathbf{b}^{(\ell)}, \quad (9)$$

where ℓ is the number of the hidden layers. Here, $\mathbf{W}^{(j)}$, $\mathbf{b}^{(j)}$, $j = 0, ..., \ell$ are the weight matrices and the bias vectors, respectively, which are conflated into the vector $\mathbf{p}$ of model parameters:

$$\mathbf{p} = \{\mathbf{W}_i^{(j)}, \mathbf{b}^{(j)} \,|\, 0 \le j \le \ell, \, 1 \le i \le n_j\}, \quad (10)$$

where n_j is the number of neurons in each layer j.

In our specific problem, $\mathbf{x} = (r, z)$ and $u : \mathbb{R}^2 \to \mathbb{R}$. Hence, the input layer of the MLP that is to approximate u, should have two neurons: one for the r coordinate and one for the z coordinate, while it has a single output neuron representing the value of u at the specific point (r, z). We experimented with various MLP architectures varying the number of hidden layers and the number of neurons per layer. Although we have not performed a rigorous optimization of the MLP, we empirically concluded that in most cases the algorithm is satisfactorily fast and accurate for an architecture with 4 hidden layers and 64 neurons per layer. This entails a total number of 12737 model parameters (weights and biases) that have to be learned. As it can be deduced by the loss history plots for various runs, the loss function for a batch of interior validation points follows closely the loss computed on the inner training points (i.e. the GGSE MSE), hence we have no indications of overfitting on the sparse inner training data (1024 points) with this particular architecture[24]. Presumably, better optimized architectures can be found, depending of course on the choice of the free functions and the free parameters of the GGSE, the complexity of the boundary and the number of training points.

Regarding the activation function, various choices have been considered, including the $tanh(x)$, the sigmoid function $\sigma(x)$, the trigonometric activation $sin(x)$ and the recently proposed *swish* function[25]: $swish(x) = x\sigma(\gamma x)$, i.e.,

$$swish(x) = \frac{x}{1 - e^{-\gamma x}}. \quad (11)$$

For $\gamma = 1$ Eq. (11) is also known as the Sigmoid-weighted Linear Unit (SiLU) activation function. In our implementation we treated γ as a learnable parameter initializing it to unity and then letting the optimizer to adjust its value in each iteration (epoch). The *swish* function has similarities with the infamous Rectified Linear Unit (ReLU) function, e.g., is bounded below and unbounded above, but unlike ReLU, it is smooth and its derivative is not a step function, so it can be used to MLPs approximating smooth solutions u.

2. Forward pass and Automatic Differentiation

Acting with the elliptic Shafranov operator on the neural network solution u_n requires the computation of partial derivatives of u_n with respect to the coordinates r, z. PINNs use Automatic Differentiation (AD)[12], that requires a forward evaluation, which is straightforward in view of (9) and then a backward pass to compute the partial derivatives, applying the chain rule to differentiate u_n with respect to the inputs $\mathbf{x}$. Thus, we understand that the compositional structure of the neural networks allows us to efficiently and accurately compute derivatives without the truncation errors that are unavoidably involved

in numerical differentiation. The same method is used for evaluating the network's derivatives with respect to the parameters **p**, that are required for the gradient-based optimization.

III. IMPLEMENTATION

A. Loss functions

In our implementation we carried out automatic differentiation and optimization using the PyTorch library[26]. In PyTorch AD is an automated procedure provided by the automatic differentiation engine **autograd**. We should note here that our approach has a lot of similarities with Ref. 22, essentially extending it to a mesh-free implementation and increasing the flexibility, e.g., regarding the choice of activation functions, the definition of the computational domain and the distribution of the training points.

The loss function that is to be minimized consists of two terms, one that measures the PDE residual error and one penalty term for the imposition of the boundary condition. These terms are:

$$L_i = \frac{1}{N_i} \sum_{j=1}^{N_i} \left| \mathcal{R}(r_j, z_j, u_n(r_j, z_j)) \right|^2 , \ (r_j, z_j) \in \mathcal{D} \quad (12)$$

$$L_b = \frac{1}{N_b} \sum_{j=1}^{N_b} \left| u_n(r_j, z_j) - u_b \right|^2 , \ (r_j, z_j) \in \partial\mathcal{D} \quad (13)$$

$$L = \lambda_i L_i + \lambda_b L_b . \quad (14)$$

In our runs we chose $N_i = N_b = N$ and we observed that the magnetic surfaces conform with the imposed boundary for $\lambda_b > \lambda_i$; otherwise, the boundary error could not be minimized sufficiently and as a consequence the computed boundary did not coincide with the imposed one. For the training process we applied two different approaches, the first was to update the learnable parameters twice in each epoch (iteration) first working through the boundary points and then through the inner points, which can be considered as a mini-batch Gradient Descend optimization with batch size equal to N from a training set consisting of $2N$ points. The second approach was a full batch Gradient descent optimization, backpropagating the error and updating the parameters once per epoch. Both implementations had similar outcomes.

B. Training data

In this study the plasma domain and its boundary are parametrically prescribed upon using appropriate analytic formulas. We investigate two different cases, one with an up-down symmetric, smooth D-Shaped boundary and the case of up-down asymmetric boundary with

a lower X-point topology. In this section we present the former case which is also employed in the next section for benchmarking the neural network solutions against analytic ones. For the up-down symmetric configuration the computational domain can be defined by the following parametric equations[27]

$$r(s,t) = 1 + \epsilon \, \xi(s) \, cos(t + \alpha sin(t)) \, ,$$
$$z(s,t) = \kappa \epsilon \, \xi(s) \, sin(t) \, ,$$
$$0 \le t \le 2\pi \, , \ 0 < s \le 1 \, , \quad (15)$$

where κ is the vertical elongation of the torus, $\epsilon = a_0/R_0$ is the inverse aspect ratio (a_0 and R_0 are the minor and major radius of the torus, respectively), and $\alpha := arcsin(\delta)$, where δ is the triangularity of the boundary. In (15) $\xi(s)$ is a monotonic function with the property $\xi(0) = 0$ and $\xi(1) = 1$. In this work we use $\xi(s) = s^k$, where $0 < k < 1$.

We create random point coordinates through Eq. (15) within the domain $\mathcal{D}$ by generating random samples for the parameters s and t from a uniform distribution over $[0, 1)$ and $[0, 2\pi)$ respectively. This results in a random distribution of points dispersed across the computational domain. Of course, one may create uniform point distributions using Eq. (15), but we prefer to use random sets so as to avoid any geometric bias in the training process. With the parameter k in $\xi(s) = s^k$ we adjust the radial density of the data points. For $k = 0.5$ we have a nearly uniform density, while for $k < 0.5$ and $k > 0.5$ the density is higher near the boundary and near the center of the parametric domain, respectively. For the boundary points we set $s = 1$ and let t take either uniformly increasing or random values in $[0, 2\pi)$. In Sec. IV we construct several equilibria for up-down symmetric and asymmetric (diverted with lower X-point) configurations with ITER-relevant geometric characteristics, i.e., $\epsilon = 2/6.2$, $\delta = 0.4$ and $\kappa = 1.6$. The sampling method for the diverted configurations with lower X-point is described in IV D.

C. Equilibrium Parametrization

Equation (1) contains three free functions that are related to the poloidal current density, the static pressure and the radial electric field. Here these free functions are represented as truncated series expansions as follows:

$$\frac{1}{2}\frac{X^2}{1 - M_p^2} = X_0 + \sum_{n=1}^{T} \frac{1}{n}X_n u^n \, ,$$

$$P_s = P_0 + \sum_{n=1}^{T} \frac{1}{n}P_n u^n$$

$$\frac{1}{2}\rho\left(\Phi'\right) = G_0 + \sum_{n=1}^{T} \frac{1}{n}G_n u^n \, , \quad (16)$$

where X_n, P_n, G_n are free parameters. In the following section we examine the cases $T = 1$, $T = 2$ and $T = 3$,

adjusting the free parameters so as to obtain realistic values of the various physical quantities of interest but not necessarily realistic profiles. We aim to investigate the possibility to approximate realistic profiles upon treating more free parameters as learnable ones in a future work.

IV. RESULTS

A. Benchmarking against linear analytic solutions

1. Solovev ansatz

For $T = 1$ in (16) we obtain the so-called Solovev ansatz[8], where the homogeneous part of the GGSE residual consists of the Shafranov differential operator acting on u, i.e., $\Delta^* u$, and the inhomogeneous part is $X_1 + P_1 r^2 + G_1 r^4$. In a previous work[9] we have derived an analytic solution to the Solovev-linearized GGSE and constructed tokamak equilibria with D-shaped cross sections with and without a lower X-point. This analytic solution is given by

$$u^*(r,z) = r \sum_j \left[a_j J_1(jr)e^{jz} + b_j J_1(jr)e^{-jz} \right.$$

$$+ c_j Y_1(jr)e^{jz} + d_j Y_1(jr)e^{-jz} \right]$$

$$- \frac{X_1}{2}z^2 - \frac{P_1}{8}r^4 - \frac{G_1}{24}r^6, \quad (17)$$

where J_1 and Y_1 are the first-order Bessel functions of first and second kind, respectively and a_j, b_j, c_j, d_j are arbitrary constants that are determined according to the desired boundary shaping.

We utilized this analytic solution to benchmark the accuracy of the resulting neural network solution. The training was performed on 1024 randomly allocated points in the domain $\mathcal{D}$ and 1024 points on the D-Shaped boundary $\partial\mathcal{D}$. The poloidal cross section of the magnetic surfaces is shown in Fig. 1 along with the training history of the corresponding neural network solution. A loss function computed on a set of inner validation points has been incorporated in the same plot. It can be seen that the validation loss is lower than the training loss (green line), hence there is no overfitting on the inner training points.

In Fig. 2 we compare the neural network solution with the solution (17). The relative error, which is defined by

$$e = \frac{|u_n - u^*|^2}{|u_a|^2}, \quad (18)$$

where u_a is the value of u_n on the magnetic axis, is of the order of 10^{-6} after 10000 epochs. In Fig. 2 we observe that the error is rather insignificant in the greater part of the cross section, while it attains larger values in certain regions near the boundary. A comparison with the analytic solution on the equatorial ($z = 0$) plane is also
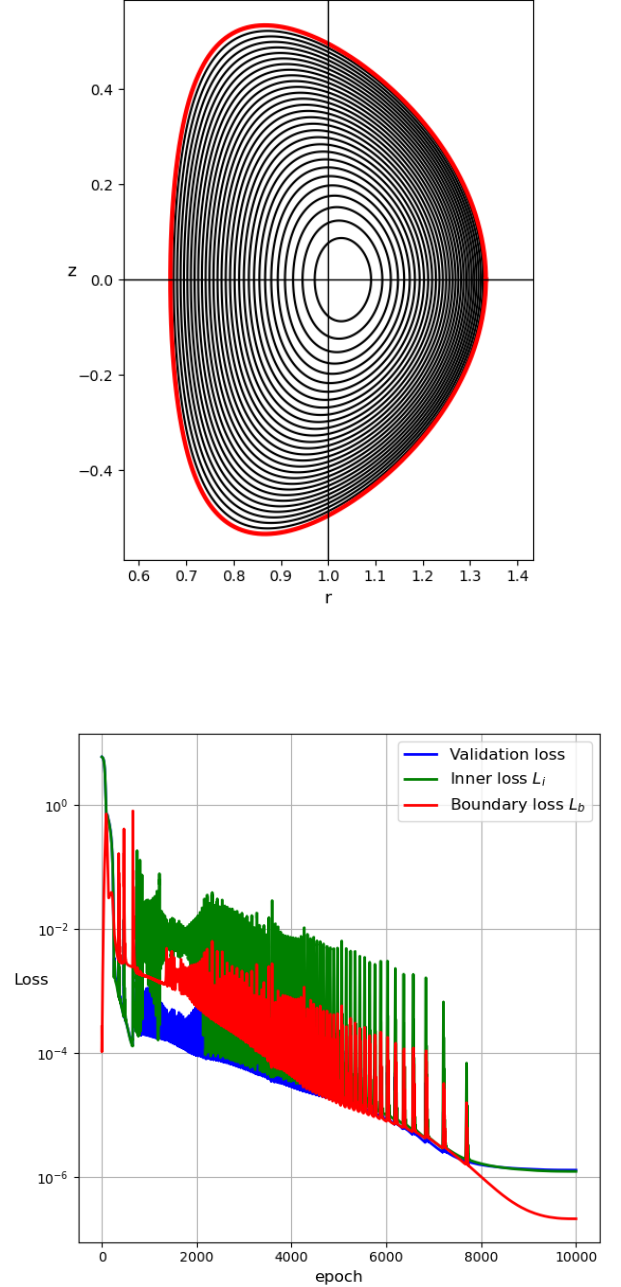

FIG. 1. Magnetic configuration (top) and training history (bottom) of the Solovev Neural Network solution

presented confirming the remarkable accuracy of the neural network solution. We performed similar comparisons for various values of free parameters and in most cases the agreement between the two solutions was similar to the particular example presented here.
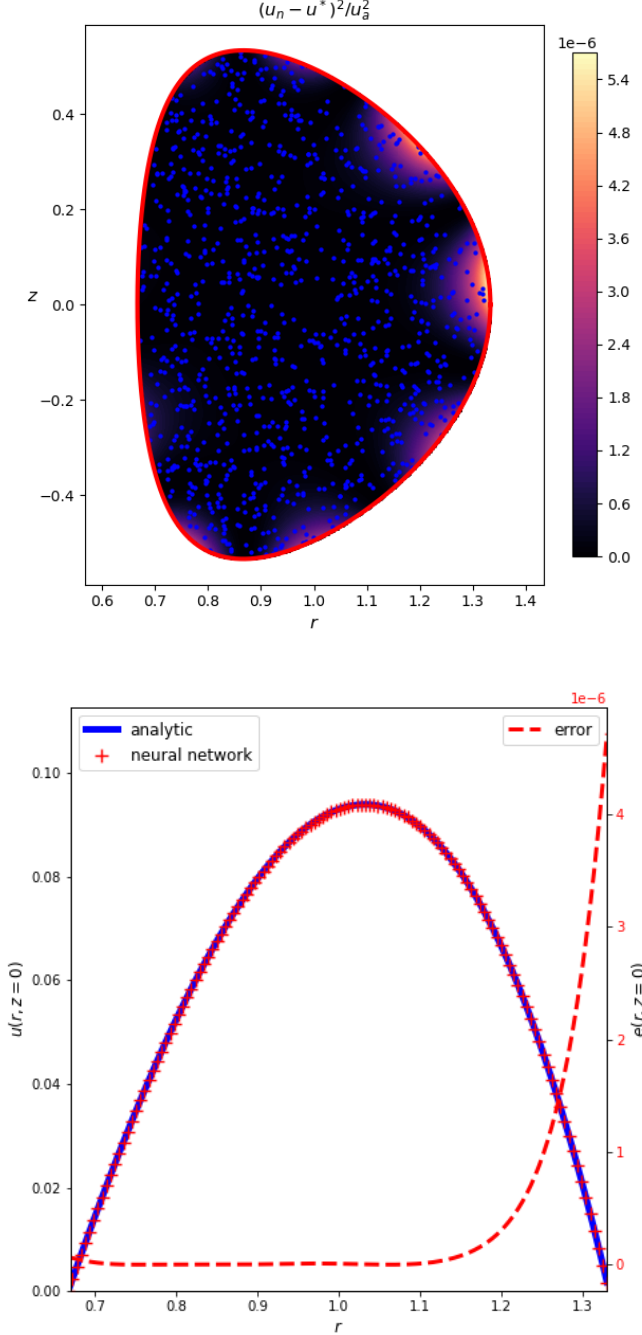
FIG. 2. Top: estimated error of the Solovev neural network solution compared with the analytic solution (17). The distribution of the randomly sampled training points (blue dots) is also presented. Bottom: comparison of the Solovev neural network with the analytic solution (17) on the equatorial plane $z = 0$.

2. General linearizing ansatz

For the general linearizing ansatz, given by (16) with $T = 2$, the GGSE residual by means of which the loss (12) is minimized, is given by

$$\mathcal{R} = \partial_{rr}u - (\partial_r u)/r + \partial_{zz}u + X_1 + P_1 r^2 + G_1 r^4$$
$$+ (X_2 + P_2 r^2 + G_2 r^4)u . (19)$$

For $G_1 = G_2 = 0$ analytic solutions to $\mathcal{R} = 0$ have been found in Ref. 28, while for the more general case $G_1, G_2 \neq 0$, we derived in Ref. 10 a general analytic solution of the form:

$$u^*(r, z) = \sum_{n=0}^{K_1} f_n(r) z^n + \sum_{n=0}^{K_2} a_n(r-1)^n , \quad (20)$$

where $f_n(r)$ are determined upon using the Frobenius method applied on the regular singular point $r = 0$. We showed that, keeping sufficiently large number of terms, when truncating the various power series which are involved in this analytic solution, the GGSE is satisfied up to machine precision (see Ref. 10 for details).

Here, we constructed a linear neural network equilibrium, which is computed upon minimizing the losses (12), (13). The geometric and profile parametric values that have been used, correspond to the particular analytic equilibrium of Ref. 10. The training history of the network is shown Fig. 3, while the resulting magnetic configuration is similar to the Solovev equilibrium. Also, the neural network solution is benchmarked against the analytic solution Eq. (20) (Fig. 4) as done previously for the Solovev solution. The high degree of agreement between the two solutions was confirmed for several choices of free parameters.

B. Interpolation capabilities of the neural network solutions

The training point sampling method described in III B allows us to adjust the radial distribution of the inner collocation points by adjusting the parameter k in Eq. (15). Varying k, we can investigate the impact of the radial distribution of points on the training process and on the accuracy of the Neural Network solution and assess its interpolation capabilities in extended regions with no collocation points.

Here we compare a neural network trained on collocation points generated by (15) with $k = 0.10$, with the neural network solution of the previous subsection where $k = 0.50$. The two training sets are superimposed in Fig. 5; a comparison between the two solutions and an estimation of the relative error is seen in Fig. 6. The maximum relative error is of the order of 10^{-5} after 10000 epochs. This result indicates that the neural network solutions can accurately describe plasma equilibria all over the computational domain even if the training is performed at its edge region.
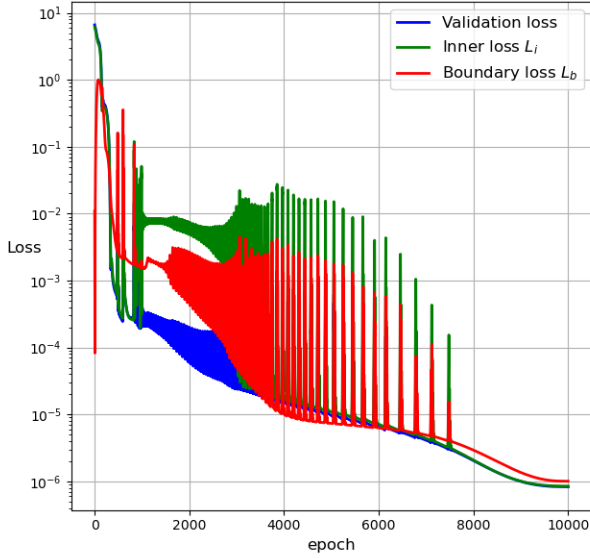
FIG. 3. Train history for the linear neural network equilibrium solution.

C. Nonlinear Equilibrium with H-mode characteristics

A nonlinear GGSE is obtained for $T > 2$. Although we present only the case $T = 3$, we have observed that the MLP algorithm performs well also for $T > 3$, depending, however, on the relative strength of the nonlinear terms. For strong nonlinear contributions the algorithm requires appropriately pre-trained networks to be used as initial conditions so as to ensure convergence.

In the particular case $T = 3$ we have:

$$\mathcal{R} = \partial_{rr}u - (\partial_r u)/r + \partial_{zz}u + X_1 + P_1 r^2 + G_1 r^4$$
$$+(X_2 + P_2 r^2 + G_2 r^4)u + (X_3 + P_3 r^2 + G_3 r^4)u^2 \quad (21)$$

1. Constrained minimization

To facilitate the parameter tuning we let P_2 and X_2 to be learned by the training process, upon requiring the average β and the total toroidal current to take specific, predetermined values, namely

$$\langle \beta \rangle = 0.03$$
$$I_t = 15\,MA\,,$$

These constraints are imposed by incorporating the following penalty terms in the total loss function

$$L_{<\beta>} = \left| \frac{1}{N} \sum_{j=1}^{N} P(r_j, u_n(r_j, z_j)) - \langle \beta \rangle \right|^2, \quad (22)$$
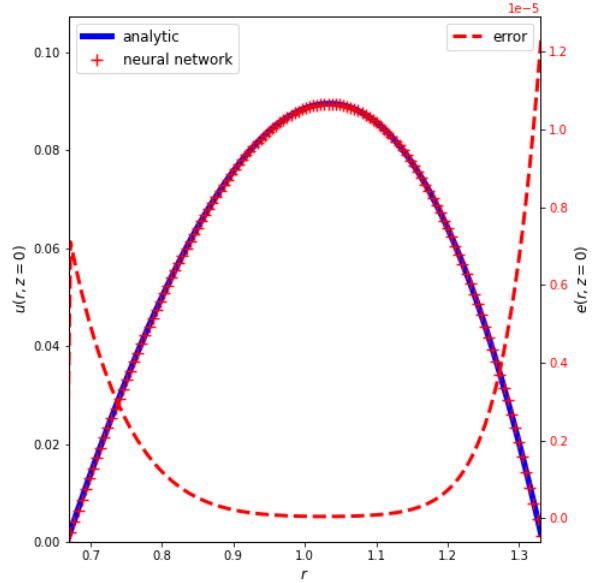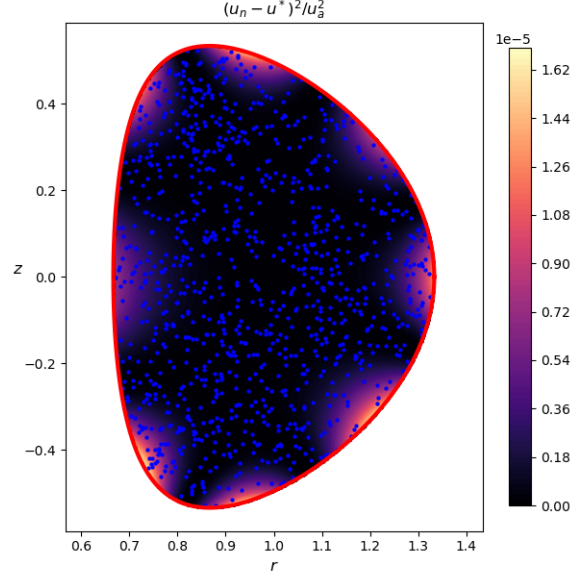

FIG. 4. Top: estimated error between the neural network and the analytic solution of the form (20) in conjuction with the randomly sampled training points (blue points). Bottom: comparison of the linear neural network and the analytic solution on the equatorial plane $z = 0$.

$$L_I = \left| \frac{S}{N} \sum_{j=1}^{N} J_\phi(r_j, u_n(r_j, z_j)) - I_t \frac{\mu_0}{R_0 B_0} \right|^2, \quad (23)$$

where, S is the poloidal cross-sectional area, P is the Alfvén-normalized dynamic pressure and

$$J_\phi = \frac{1}{r[1 - M_p^2(u)]^{1/2}} \left(\Delta^* u + \frac{M_p(u)M_p'(u)}{1 - M_p^2(u)} |\nabla u|^2 \right) \quad (24)$$
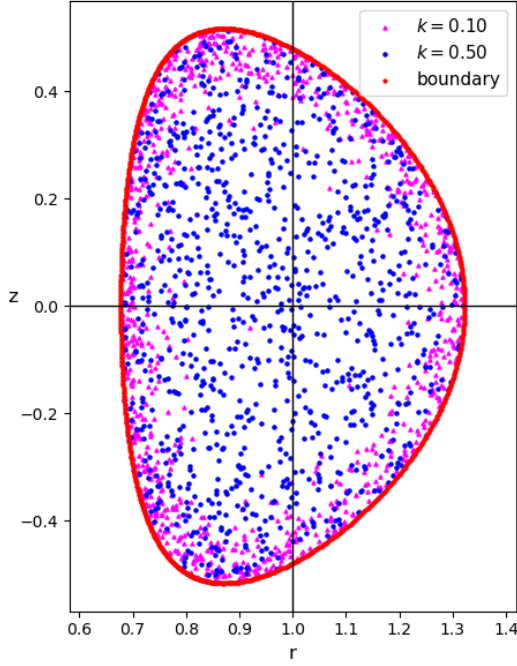
is the normalized toroidal current density.

FIG. 5. Distribution of training points, for $k = 0.50$ (blue dots) and $k = 0.10$ (magenta triangles). The boundary points (red dots) are uniformly distributed in both cases.

2. H-mode equilibrium

The mechanisms behind the L-H transition and the associated phenomenology, e.g., the transport barrier and density pedestal formation, is not yet fully understood even from the force balance point of view, although since its discovery in 1982[29], most tokamaks are designed to operate in H-mode. Therefore, the construction of equilibrium states encompassing H-mode characteristics such as density and pressure pedestals (see, for example, the recent works 10, 30–32) is important for fusion research. Moreover, the construction of such states might be useful for performing plasma stability and transport studies concerning the H-mode operational regime.

An H-mode-pertinent ansatz[33] that results in a mass density pedestal is given by

$$\rho(u) = (\rho_1 + \rho_2 u^2)\left(1 - \exp\left(-u^2/\mu\right)\right), \quad (25)$$

while for the Mach function $M_p(u)$ we choose a shifted Gaussian[30], i.e.

$$M_p(u) = M_0 \exp\left(-\frac{(u - u_f)^2}{2\sigma^2}\right), \quad (26)$$

where M_0 is the maximum Mach number, u_f is the value of u on the magnetic surface where M_p is maximum and
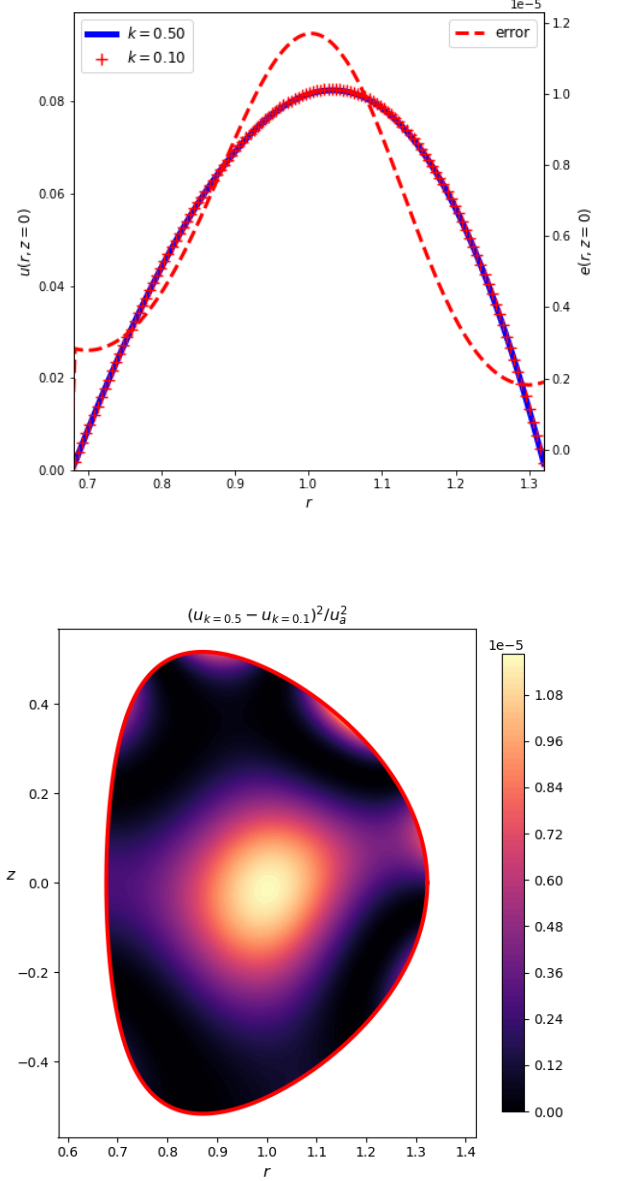

FIG. 6. Comparison between two neural network solutions trained on points with different radial distributions, on the equatorial plane (top) and their relative error (bottom).

σ is the width of the Gaussian profile. Using these free functions and the neural network solutions for the nonlinear GGSE we define a tokamak equilibrium encompassing some H-mode characteristics. In Fig. 8 we present the mass density profile that exhibits a characteristic pedestal morphology and the computed toroidal velocity profile. In Fig. 9 the pressure and the rotational to kinetic energy density ratio profiles are provided while in Fig. 10 we present the toroidal current density and the safety factor profile. Using the general linear solution
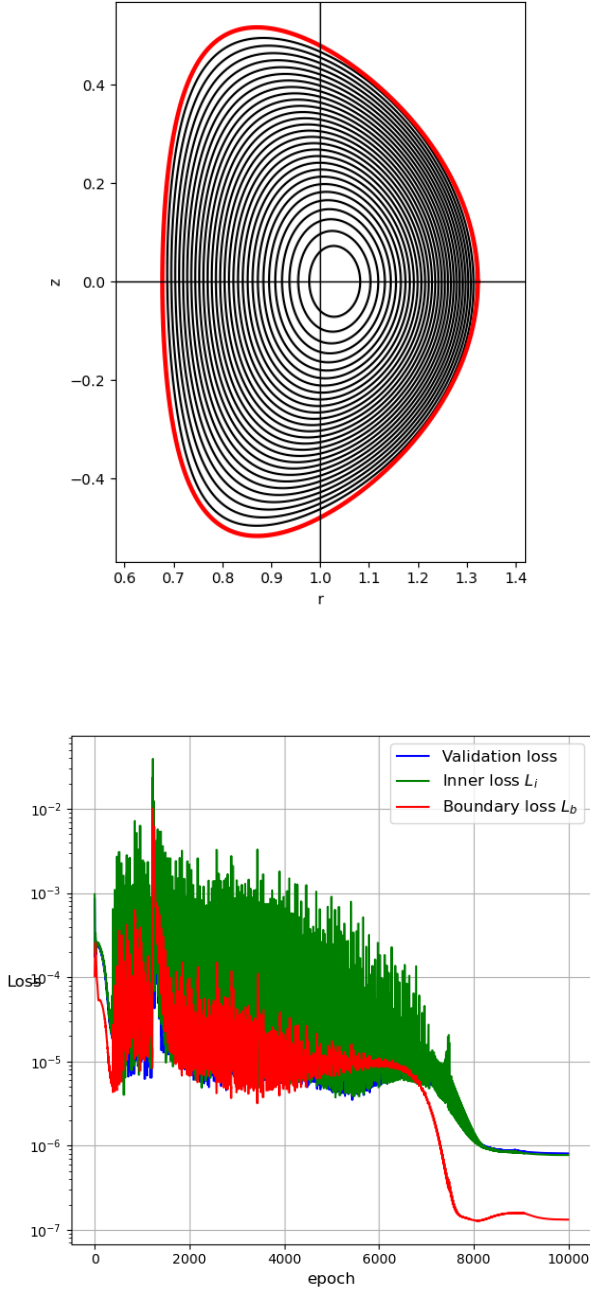
FIG. 8. The mass density (top), and the toroidal velocity profile (bottom) plotted on the midplane $z = 0$.

FIG. 7. The magnetic field configuration for the nonlinear ansatz $(T = 3)$ (top) and the corresponding training history of the neural network solution (bottom).

we obtain similar profiles (not presented here) with noticeable but not major differences. Significant differences are observed upon increasing the parameters P_3 and X_3, however, in this case though the convergence deteriorates when, in addition, the constraints (22) and (23) are preserved.
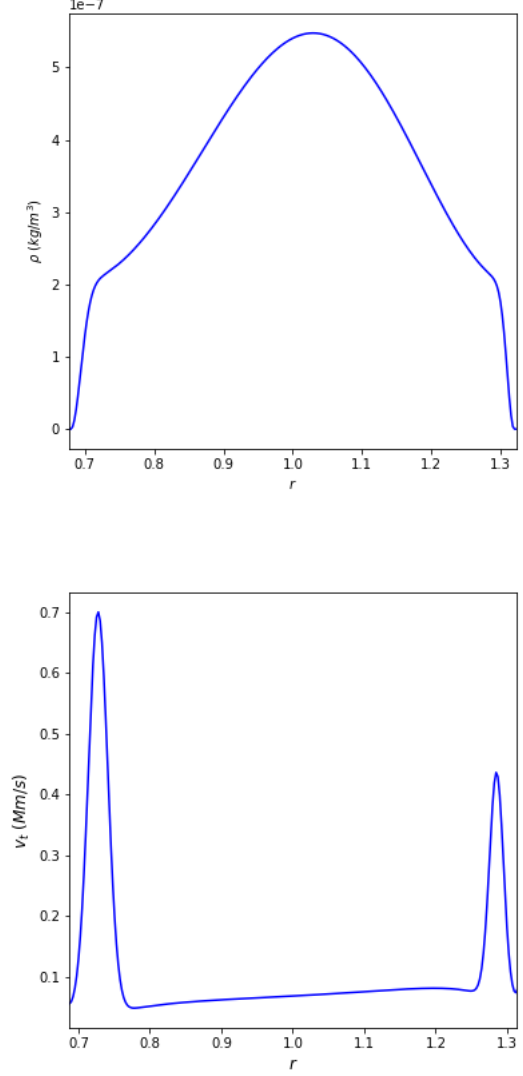
The toroidal velocity is given by the following expression

$$v_\phi = \frac{M_p}{r\sqrt{\rho}}I - r\Phi'(u)\sqrt{1 - M_p^2}\,, \qquad (27)$$

while the plasma pressure can be computed by a Bernoulli equation[5]

$$P = P_s(u) - \rho\left\{\frac{v^2}{2} - r^2\left[\Phi'(u)\right]^2\right\}\,, \qquad (28)$$

where $v^2 = \sqrt{v_\phi^2 + v_p^2}$, with

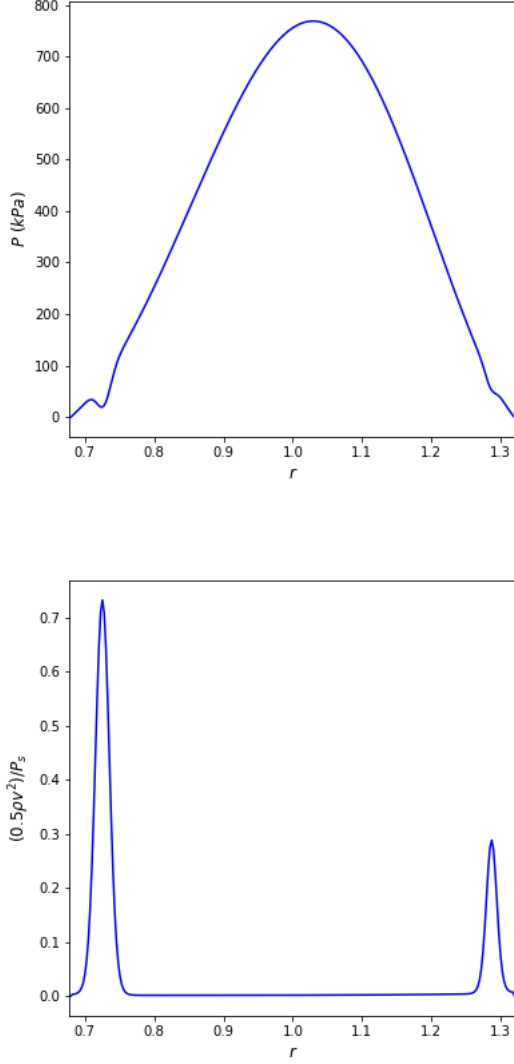$$v_p^2 = \left(\frac{M_p}{r\sqrt{\rho}}\right)^2 |\nabla u|^2\,.$$

FIG. 9. The total pressure profile (top) and the rotational to kinetic energy density ratio (bottom), plotted against the cylindrical coordinate r. The rotational to kinetic energy ratio is 0.02.

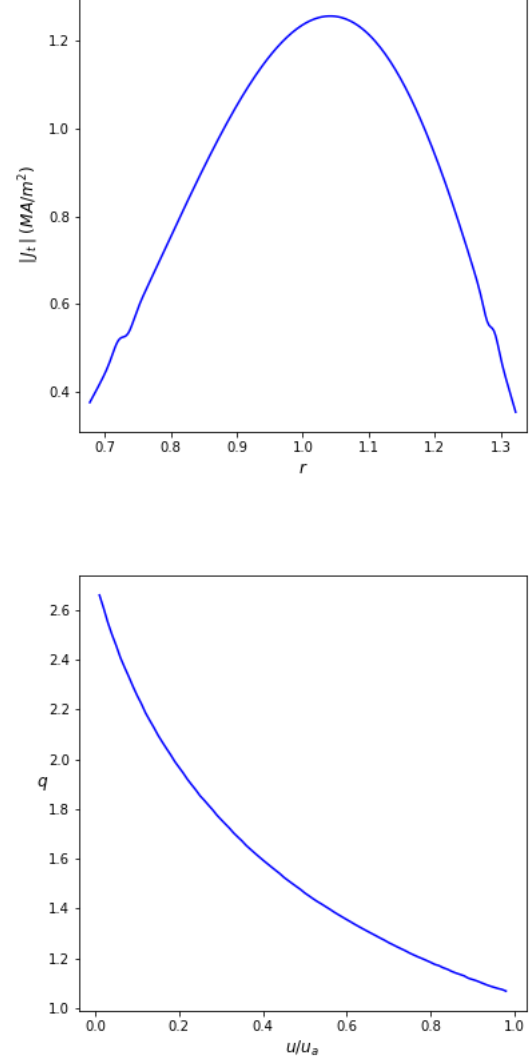FIG. 10. The toroidal current density profile (top) plotted on the midplane $z = 0$ and he safety factor (bottom) plotted against u/u_a.

D. Equilibrium with lower X-point

For a diverted configuration with boundary curve forming a lower X-point, the following parametric equations[34] are used to define the computational domain:

$$r(s,t) = \begin{cases} 1 + \epsilon\,\xi(s)\,cos(t + \alpha sin(t))\,, & 0 \le t \le \pi\,, \\ 1 + \epsilon\xi(s)cos(t)\,, & \pi \le t \le 2\pi\,, \end{cases} \tag{29}$$

$$z(s,t) = \begin{cases} \kappa\epsilon\,\xi(s)\,sin(t)\,, & 0 \le t \le \pi\,, \\ -\kappa_d\epsilon \left[\frac{\xi(s)(1+cos(t))}{1+cos(\theta_d)}\right]^{1/2}\,, & \pi \le t < 2\pi - \theta_d\,, \\ -\kappa_d\epsilon \left[\frac{\xi(s)(1-cos(t))}{1+cos(\theta_d)}\right]^{1/2}\,, & 2\pi - \theta_d \le t \le 2\pi\,, \end{cases}$$

where $0 \le s \le 1$ and $\theta_d = \pi - arctan(\kappa_d/\delta_d)$.

For the X-point equilibrium we had to double the number of collocation points and epochs in order to obtain a magnetic configuration with the desired geometric characteristics. Namely, $N_i = N_b = 2048$ and the neural network has been trained for 20000 epochs. The resulting magnetic configuration and the corresponding training history of the neural network are presented in Fig. 11.
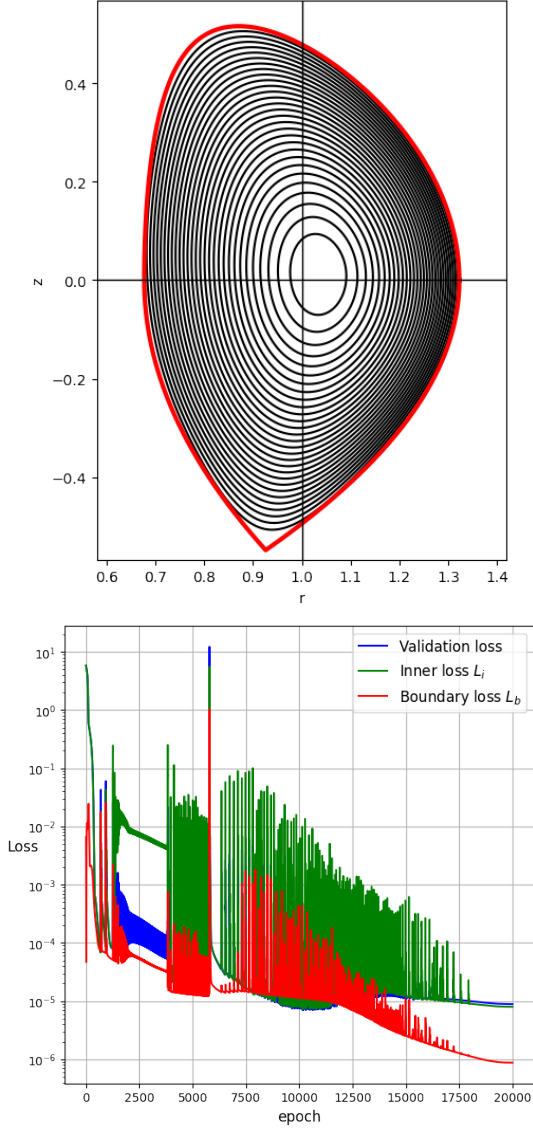
---

upon localizing the distribution of the training points near the plasma edge. The results presented here will be further corroborated upon benchmarking against tested numerical equilibrium solvers such as HELENA and considering realistic equilibrium profiles in a future work. Extensive comparisons in terms of the attained accuracy and flexibility of solutions in conjuction with the corresponding CPU time will also be attempted[35].

FIG. 11. Equilibrium configuration with lower X-point in connection with the neural network solution of the general linearized GGSE (top). The training history of the neural network equilibrium with lower X-point (bottom).

V. CONCLUSIONS

In this work we computed neural network solutions to a generalized Grad-Shafranov equation (GGSE), governing axisymmetric MHD equilibria with incompressible flows of arbitrary direction. The neural networks were trained in domains with tokamak-pertinent shapes adopting linear and nonlinear choices for the free functions which are involved in the GGSE. The solutions were benchmarked against exact, analytic solutions derived in previous works, displaying fairly good agreement. In addition, the remarkable capability of the neural networks to approximate equilibrium solutions in regions with no collocation points was demonstrated

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

[1]V. D. Shafranov, Sov. Phys. JETP **6**, 545 (1958).
[2]H. Grad and H. Rubin, Proceedings of the Second United Nations Conference on the Peaceful Uses of Atomic Energy (United Nations, Geneva, 1958) **Vol. 31**, p. 190 (1958).
[3]L. Guazzotto, R. Betti, J. Manickam, and S. Kaye, "Numerical study of tokamak equilibria with arbitrary flow," Physics of Plasmas **11**, 604–614 (2004).
[4]K. H. Burrell, "Effects of e×b velocity shear and magnetic shear on turbulence and transport in magnetic confinement devices," Physics of Plasmas **4**, 1499–1518 (1997).
[5]H. Tasso and G. N. Throumoulopoulos, "Axisymmetric ideal magnetohydrodynamic equilibria with incompressible flows," Physics of Plasmas **5**, 2378–2383 (1998).
[6]C. Simintzis, G. N. Throumoulopoulos, G. Pantis, and H. Tasso, "Analytic magnetohydrodynamic equilibria of a magnetically confined plasma with sheared flows," Physics of Plasmas **8**, 2641–2648 (2001).
[7]P. Morrison, "$\delta^2 f$: A generalized energy principle for determining linear and nonlinear stability." Bull. Am. Phys. Soc. **31**, 1609 (1986).
[8]L. S. Solovév, Sov. Phys. JETP **26**, 400 (1968).
[9]D. A. Kaltsas and G. N. Throumoulopoulos, "Generalized solovev equilibrium with sheared flow of arbitrary direction and stability consideration," Physics of Plasmas **21**, 084502 (2014).
[10]D. A. Kaltsas, A. Kuiroukidis, and G. N. Throumoulopoulos, "A tokamak pertinent analytic equilibrium with plasma flow of arbitrary direction," Physics of Plasmas **26**, 124501 (2019).
[11]A. J. Cerfon and J. P. Freidberg, ""one size fits all" analytic solutions to the grad–shafranov equation," Physics of Plasmas **17**, 032502 (2010).

[12] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," J. Mach. Learn. Res. **18**, 5595–5637 (2017).

[13] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," Journal of Computational Physics **378**, 686–707 (2019).

[14] B. P. van Milligen, V. Tribaldos, and J. A. Jiménez, "Neural network differential equation and plasma equilibrium solver," Phys. Rev. Lett. **75**, 3594–3597 (1995).

[15] S. Joung, J. Kim, S. Kwak, J. Bak, S. Lee, H. Han, H. Kim, G. Lee, D. Kwon, and Y.-C. Ghim, "Deep neural network grad–shafranov solver constrained with measured magnetic signals," Nuclear Fusion **60**, 016034 (2019).

[16] V. Tribaldos and B. P. van Milligen, "Neural network tool for rapid recovery of plasma topology," Review of Scientific Instruments **68**, 931–934 (1997).

[17] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," IEEE Transactions on Neural Networks **9**, 987–1000 (1998).

[18] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural Networks **2**, 359–366 (1989).

[19] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "Deepxde: A deep learning library for solving differential equations," SIAM Review **63**, 208–228 (2021).

[20] F. Chen, D. Sondak, P. Protopapas, M. Mattheakis, S. Liu, D. Agarwal, and M. D. Giovanni, "Neurodiffeq: A python package for solving differential equations with neural networks," Journal of Open Source Software **5**, 1931 (2020).

[21] A. Koryagin, R. Khudorozkov, and S. Tsimfer, "Pydens: A python framework for solving differential equations with neural networks," arXiv:1909.11544 (2019).

[22] J. B. Pedro, J. Maronas, and R. Paredes, "Solving partial differential equations with neural networks," ArXiv (2019).

[23] J. Blechschmidt and O. G. Ernst, "Three ways to solve partial differential equations with neural networks — a review," GAMM-Mitteilungen **44**, e202100006.

[24] A set of validation points can be constructed by dividing the original batch of inner points in two sub-batches. The one is then used for training the network and the other for assessing its performance.

[25] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," (2017), arXiv:1710.05941 [cs.NE].

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in Advances in Neural Information Processing Systems 32, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc., 2019) pp. 8024–8035.

[27] A. D. Turnbull, Y. R. Lin-Liu, R. L. Miller, T. S. Taylor, and T. N. Todd, "Improved magnetohydrodynamic stability through optimization of higher order moments in cross-section shape of tokamaks," Physics of Plasmas **6**, 1113–1116 (1999).

[28] C. V. Atanasiu, S. Günter, K. Lackner, and I. G. Miron, "Analytical solutions to the grad–shafranov equation," Physics of Plasmas **11**, 3510–3518 (2004).

[29] F. Wagner, "A quarter-century of h-mode studies," Plasma Physics and Controlled Fusion **49**, B1–B33 (2007).

[30] H. Li and P. Zhu, "Formation of edge pressure pedestal and reversed magnetic shear due to toroidal rotation in a tokamak equilibrium," Physics of Plasmas **28**, 054505 (2021).

[31] L. Guazzotto and J. P. Freidberg, "Simple, general, realistic, robust, analytic tokamak equilibria. part 2. pedestals and flow," Journal of Plasma Physics **87**, 905870305 (2021).

[32] G. Montani and M. D. Prete, "Axisymmetric plasma equilibria with toroidal and poloidal velocity fields: Tokamak relevant configurations," (2021), arXiv:2107.14766 [physics.plasm-ph].

[33] A. Pataki, A. Cerfon, J. Freidberg, L. Greengard, and M. O'Neil, "A fast, high-order solver for the grad-shafranov equation," J. Comput. Phys. **243**, 28–45 (2013).

[34] A. Kuiroukidis and G. N. Throumoulopoulos, "Comment on the paper 'an analytic functional form for characterization and generation of axisymmetric plasma boundaries'," Plasma Physics and Controlled Fusion **57**, 078001 (2015).

[35] To provide a quantitative measure of CPU time, we executed the code in a personal computer with an Intel i7-7500U CPU at 2.70GHz (2 cores) without GPU acceleration. For the nonlinear example of Sec. IV with 2048 training points and $12737+3$ learnable parameters, each epoch takes approximately 0.1 s, while the entire runtime is approximately 17 minutes.