Machine learning surrogate models for Landau fluid closure

Cite as: Phys. Plasmas **27**, 042502 (2020); https://doi.org/10.1063/1.5129158
Submitted: 26 September 2019 . Accepted: 28 February 2020 . Published Online: 01 April 2020

Chenhao Ma, Ben Zhu, Xue-Qiao Xu, and Weixing Wang

NEW
AVS Quantum Science
A new interdisciplinary home for impactful quantum science research and reviews
Co-Published by
NOW ONLINE

Machine learning surrogate models for Landau fluid closure

Cite as: Phys. Plasmas **27**, 042502 (2020); doi: 10.1063/1.5129158
Submitted: 26 September 2019 · Accepted: 28 February 2020 ·
Published Online: 1 April 2020

Chenhao Ma,[1,a)] Ben Zhu,[2,b)] Xue-Qiao Xu,[2] and Weixing Wang[1]

AFFILIATIONS

[1]Theory Department, Princeton Plasma Physics Laboratory, Princeton, New Jersey 08540, USA
[2]Fusion Energy Science Program, Lawrence Livermore National Laboratory, Livermore, California 94550, USA

[a)]Author to whom correspondence should be addressed: cma@pppl.gov
[b)]Electronic mail: zhu12@llnl.gov

ABSTRACT

The first result of applying the machine/deep learning technique to the fluid closure problem is presented in this paper. As a start, three different types of neural networks [multilayer perceptron (MLP), convolutional neural network (CNN), and two-layer discrete Fourier transform (DFT) network] were constructed and trained to learn the well-known Hammett–Perkins Landau fluid closure in configuration space. We find that in order to train a well-preformed network, a minimum size of the training data set is needed; MLP also requires a minimum number of neurons in the hidden layers that equals the degrees of freedom in Fourier space, despite the fact that training data are being fed into the configuration space. Out of the three models, DFT performs the best for the clean data, most likely due to the existence of the simple Fourier expression for the Hammett–Perkins closure, but it is the least robust with respect to input noise. Overall, with appropriate tuning and optimization, all three neural networks are able to accurately predict the Hammett–Perkins closure and reproduce the intrinsic *nonlocal* feature, suggesting a promising path to calculating more sophisticated closures with the machine/deep learning technique.

I. INTRODUCTION

The fluid closure problem in plasma physics is probably as old as plasma physics itself. The problem arises when deriving fluid equations through the chains of moment equations from kinetic theories. The resulting lower order moment equations always contain a higher order moment. To truncate the moment hierarchy, a proper *closure* is thus required to approximate this higher order moment from existing lower order moments for microscopic descriptions, which is conventionally constructed by phenomenological constitutive relations. Moment closure hierarchies for kinetic theories are important, and active scientific areas of research include fluid dynamics, plasma physics, neuroscience, and so on. The widely used Spitzer–Härm closure[1] and, similarly, the Braginskii closure[2] consider a strongly collisional plasma and predict heat flux $q \propto \nabla T$, both of which lack kinetic effects and start to break down when the particle mean-free-path approaches the characteristic length scale (i.e., in the weakly collisional regime). Hammett and Perkins first proposed a so-called Landau-fluid closure as it incorporates Landau-damping effects in the electrostatic, collisionless, static limit.[3] Over the years, the Landau-fluid closure has been extended to collisional,[4,5] magnetized[6] plasma, and dynamic perturbation conditions.[7,8] However, implementing Landau-fluid closures to high performance fluid codes is numerically challenging as the closures usually involve both frequency and wave-vectors in Fourier space.[4,7,8]

Over the same period of time, there has been significant progress in the machine/deep learning method.[9] Inside the fusion plasma physics community, models have been trained to assist experimental data analysis,[10] carry out real-time modeling of neutral beam injection,[11] aid operation control (e.g., disruption prediction[12–14]) to evaluate the core turbulent transport fluxes and the pedestal structure,[15] design better ICF experiments,[16] and so on. Here, we are interested in another kind of machine learning application—generating machine learning surrogate models of phenomena for microscopic descriptions that can be used within models for fluid simulations, yielding better and faster solutions. An interesting, open question is thus raised—is it possible to get a machine-learned closure model for plasma physics? Or in other words, can a neural network be trained with existing theories, or data from simulations to learn the closure which could be further applied to fluid simulations? If the answer is affirmative, then how do we build and train such a model? How is its performance in terms of accuracy and speed-up?

As a proof of principle, we start with training neural networks to learn the Hammett–Perkins closure—it is relatively simple (thus understandable), analytically solvable (thus easy to evaluate errors), yet non-trivial as it represents certain aspects of general closure problems (e.g., the *nonlocal* feature is explained below). Furthermore, even though it is simple in Fourier space, it is difficult or impossible to implement in fluid simulation codes, such as BOUndary Turbulence (BOUT++) code,[17] and Global Drift Ballooning (GDB) model,[18] in configuration space due to particular discretizations, geometries, and boundary conditions.[19] In such circumstances, training the Hammett–Perkins closure in configuration space is desired.

II. PHYSICS MODEL

The Hammett–Perkins closure[3] gives the relation between temperature and heat flux perturbation in Fourier space as

$$\tilde{q}_k = -n_0\sqrt{\frac{8}{\pi}}v_t\frac{ik\tilde{T}_k}{|k|}. \tag{1}$$

Transforming Eq. (1) from Fourier space back to configuration space yields

$$\tilde{q}(x) = -n_0\sqrt{\frac{8}{\pi^3}}v_t\int_0^\infty dx'\left[\tilde{T}(x+x') - \tilde{T}(x-x')\right]/x'. \tag{2}$$

The convolution in Eq. (2) indicates that the heat flux $\tilde{q}(x)$ is *nonlocal* in configuration space, i.e., the perturbed heat flux $\tilde{q}(x)$ at location x relies on not only the local temperature $\tilde{T}(x)$ and its gradient $\partial\tilde{T}(x)/\partial x$, but also on the nearby temperature perturbations $\tilde{T}(x \pm x')$.

The motivation of this study is to assess the neural networks' capability of recovering the nonlocal feature. It shall be noted that in Fourier space, the expression of the Hammett–Perkins closure is linear and local, thus training a neural network for Eq. (1) becomes trivial. Therefore, in this paper, we focus on training neural networks to learn Hammett–Perkins closure Eq. (2) in configuration space. At the same time, we try to answer several technical questions such as: how much data is needed to train a neural network? Are there any constraints on the neural network configuration? What type of neural network performs best and why?

For simplicity, we assume that plasma has uniform density n_0, simulation domain $x \in [0, 2\pi]$, and resolution $n_x = 128$ with the periodic boundary condition. To prepare the data set for model training, validation, and testing, a series of fluctuating temperature profiles $\{\tilde{T}_i(x)\}$ are first generated with

$$\tilde{T}_i(x) = \sum_{k=1}^{k_{\max}} A_{ik}\sin(kx + \phi_{ik}), \tag{3}$$

where uniformly distributed A_{ik} and ϕ_{ik} are randomly chosen in the range $0 \le A_{ik} \le 1$ and $0 \le \phi_{ik} < 2\pi$; the corresponding $\tilde{q}_i(x)$ is then calculated with Eq. (1) by Fourier transform. Figure 1 shows one pair of $\tilde{T}(x)$ (blue) and $\tilde{q}(x)$ (red). Finally, the paired temperature and heat flux profiles $\{\tilde{T}_i(x), \tilde{q}_i(x)\}$ are re-normalized to fit in between -1 and 1,

$$\hat{\tilde{T}} = \frac{\tilde{T}}{\tilde{T}_{max}}, \quad \hat{\tilde{q}} = \frac{\tilde{q}}{\tilde{q}_{max}}, \tag{4}$$

where $\tilde{T}_{max}$ and $\tilde{q}_{max}$ are the maximum absolute values of $\tilde{T}(x)$ and $\tilde{q}(x)$ in the entire data set, respectively.

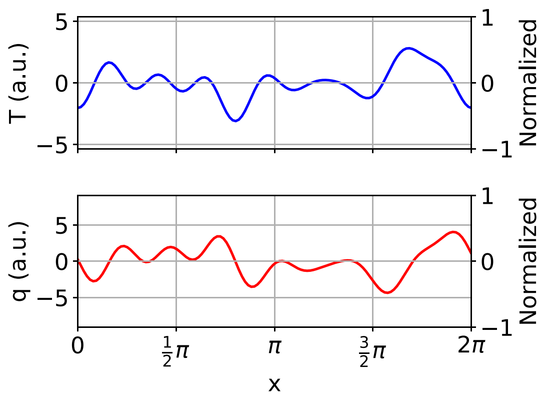

FIG. 1. Fluctuating (a) temperature and (b) heat flux profiles.

III. MACHINE LEARNING SURROGATE MODELS

In this study, three types of neural networks are constructed with the Keras interface,[20] and TensorFlow[21] backend as shown in Fig. 2. The activation and loss functions are two key concepts in neural networks. The former defines the output of a neuron given an input or a set of inputs. The rectified linear unit (ReLU) is the most commonly used activation function in deep learning, which is defined as $f(x) = \max\{x, 0\}$. The latter defines a mean to evaluate model performance by comparing the discrepancy between the estimated and true values of the solution. Two popular loss functions for the regression problem are the mean-squared-error (MSE) and the mean-absolute-error (MAE), which are defined as $\epsilon_{mse} = \sum_i|\hat{y}_i - y_i|^2/N$ and $\epsilon_{mae} = \sum_i|\hat{y}_i - y_i|/N$. Here, $\hat{y}_i$ and y_i are the estimated and true values, respectively, and N is the number of samples.

The first type of neural network we train is a conventional multi-layer perceptron (MLP).[22] It contains one input layer, two hidden layers, and one output layer. Either ReLU or the tanh function is chosen as the activation function in MLP except on the last layer where the linear function is applied. The second type of neural network is a convolutional neural network (CNN).[23] It has an input layer, double convolution-pooling layers, followed by two convolution layers, and double upsampling-convolution layers. In CNN, the activation function is ReLU for hidden layers, and linear for the output layer. The third type of neural network is a two-layer discrete Fourier transform (DFT) neural network.[24] All three networks use the standard Adam optimization algorithm,[25] and MSE as the loss function for training and validation. However, in the following discussion, we primarily use MAE and the mean of MAE distribution $\bar{\epsilon}_{mae}$ as the error metric since it is more intuitive, i.e., no square-root operation is needed to evaluate the relative error.

IV. TRAINING RESULTS

Neural networks are data-hungry, i.e., the performance of the neural networks strongly depends on the amount of training data available. A well trained neural network shall also be neither overfit nor underfit,[9] i.e., it should have a minimum discrepancy between training and validation loss. The first task is therefore trying to find out how much data is adequate for this simple 1D closure problem by

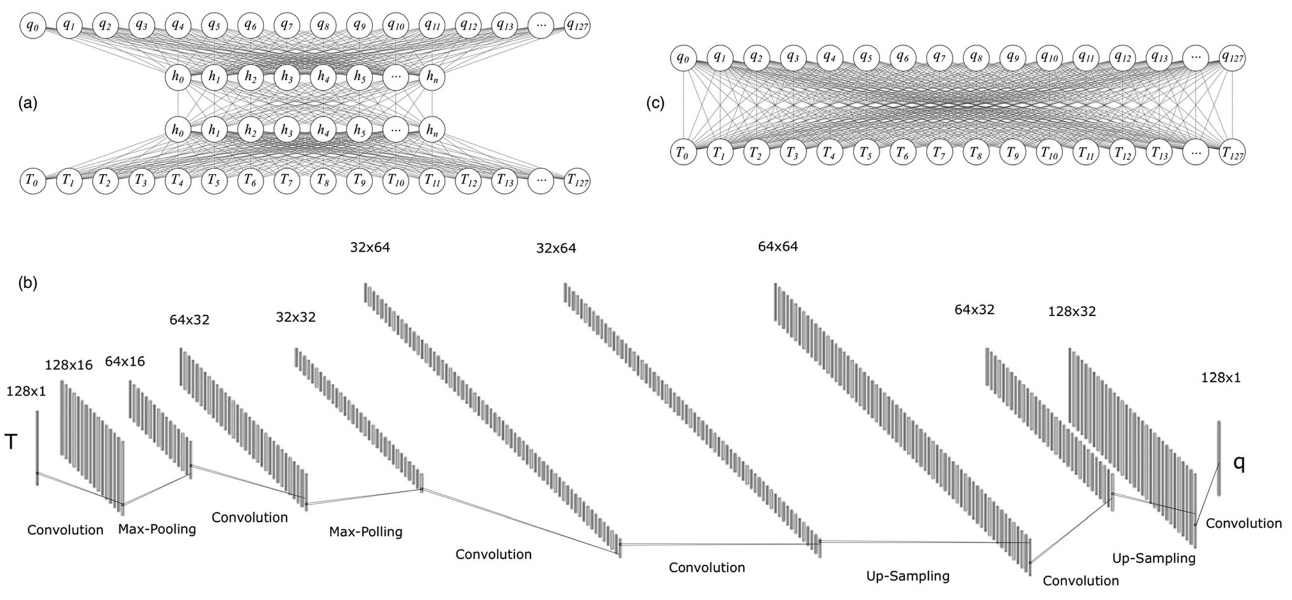

FIG. 2. Neural network architecture schematics of (a) multilayer perceptron (MLP): one input layer, two hidden layers, and one output layer with 128 input and output variables and n neurons for each hidden layer; (b) convolutional neural network (CNN): one input layer, double convolution-pooling layers, followed by two convolution layers, double upsampling-convolution layers, and one output layer with 128 input and output variables; and (c) discrete Fourier transform neural network (DFT): one input layer and one output layer with 128 input and output variables and no hidden layers.

comparing the training loss and the validation loss. We start to train the MLP model with an increasing size of training data n_{sample}.

Figure 3 illustrates the training history of MLP with $n_{\text{sample}} = 100$ and $n_{\text{sample}} = 10\,000$. Here, the "epoch" in the horizontal axis means the number of times that the entire training data set is passed to the neural network. In both cases, training and testing MAEs start at a similar value. During the training process, the training MAE decreases much faster than the testing MAE for the $n_{\text{sample}} = 100$ case; and the discrepancy between training and testing MAEs at the converged phase indicates that the MLP is overfitted, suggesting the training data set is insufficient. For the $n_{\text{sample}} = 10\,000$ case, the training

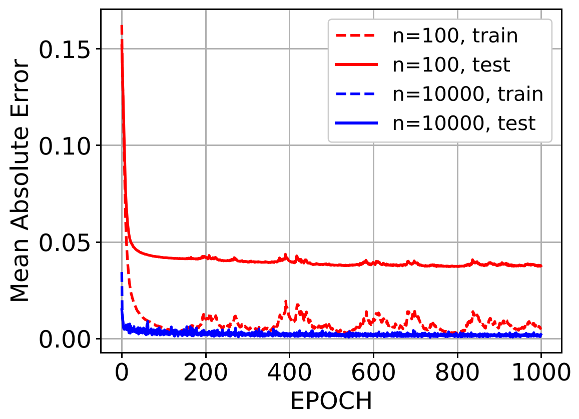

FIG. 3. Training history of MLP with different data sizes: (red) $n_{\text{sample}} = 100$ and (blue) $n_{\text{sample}} = 10\,000$. The dashed and solid lines denote the training and testing Mean-Absolute-Error, respectively.

MAE and the testing MAE have similar values almost over the whole training process. We hence conclude that at least $n_{\text{sample}} = 10\,000$ training data is required for the 1D closure problem at this resolution for MLP. Figure 3 also shows that the trained model tends to converge faster with a larger data set: when $n_{\text{sample}} = 100$, the model converges near epoch = 150; while for $n_{\text{sample}} = 10\,000$, it converges near epoch = 60.

In machine/deep learning, "hyperparameter" refers to a parameter that is chosen before the training process, e.g., the activation function selection, hidden layer configuration, optimization algorithms, and so on. Choosing the correct hyperparameter is critical for model performance. Here, we explore the impact of different activation functions. Figure 4 shows the MAE of different types of neural networks with different activation functions and different n_{sample} used in the training. In general, the discrepancy between the training MAE and the testing MAE decreases when n_{sample} increases from 10 to 1 000 000, except for MLP with $n_{\text{sample}} = 10$ when the training MAE is 6th order of magnitude smaller than the testing MAE. As discussed above, the apparent departure is an indicator that the model is either overfit or underfit. In this case, the MLP is clearly overfit as n_{sample} is too small and the resulting model poorly predicts the heat flux for the testing data. On the other hand, the near-perfect overfit with the $n_{\text{sample}} = 10$ result proves that the fully connected MLP is indeed able to capture the nonlocal effect of Hammett–Perkins closure Eq. (2). When n_{sample} increases, the training MAE increases since the model can no longer overfit all the training data. Meanwhile, the testing MAE decreases, until they converge to the same value when n_{sample} is large enough. The test results also show that the MLP with the tanh function converges with less data at $n_{\text{sample}} = 100$, and gives a smaller testing MAE than the MLP with the ReLU function when $n_{\text{sample}} \lesssim 1000$. This is because the tanh function suffers from the

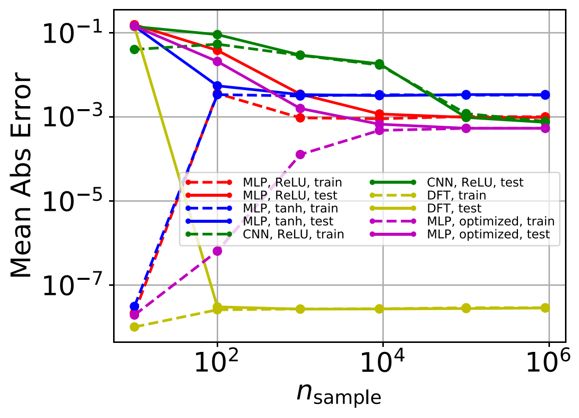

FIG. 4. Mean-Absolute-Error with the number of training samples. The dashed and solid lines denote the training and testing MAE, respectively; the red, blue, and green lines represent MLP with ReLU, MLP with tanh, and CNN with ReLU as the activation functions, respectively. The yellow line represents the DFT result. The purple line represents the result of the Bayesian optimized model.

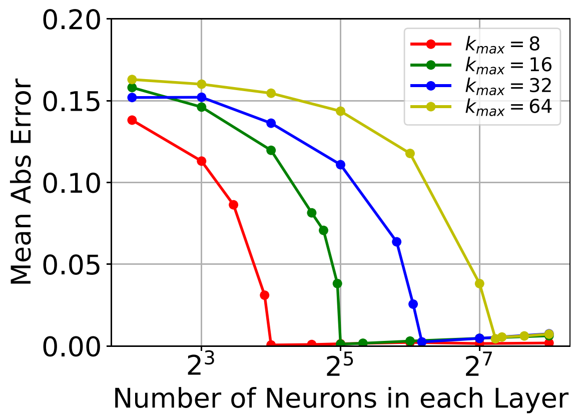

FIG. 5. Mean-Absolute-Error of MLP for different numbers of neurons in each layer. Red for $k_{\rm max} = 8$, green for $k_{\rm max} = 16$, blue for $k_{\rm max} = 32$, and yellow for $k_{\rm max} = 64$.

vanishing gradient problem which prevents the neural network from further training even with more data. Thus, the MLP with ReLU has better performance when the training data is larger (e.g., $n_{\rm sample} > 10\,000$ in our test). The purple line shows the results of the neural network which utilizes the Bayesian approach to optimize the number of layers and the number of neurons per layer for the MLP; the most accurate MLP model is found to be one hidden layer with 47 neurons, and the resulting $\epsilon_{MAE}^{\rm Bayesian} = 5.49 \times 10^{-4}$ after 100 optimization iterations. This is a 43% accuracy improvement compared to the two layer model (red line). The yellow lines show the result of the DFT network, which has a testing MAE $\epsilon_{DFT} = 2.69 \times 10^{-8}$. This is because it represents a linear matrix multiplication. The DFT network will be discussed later.

The width and depth of neural networks (i.e., the number of neurons in the hidden layers $n_{\rm neuron}$ and the number of hidden layers) are also crucial hyperparameters in a neural network. Here, we first test the impact of $n_{\rm neuron}$ on the testing MAE for a two hidden layer MLP. Figure 5 shows that the testing MAE varies with $n_{\rm neuron}$ for different numbers of modes $k_{\rm max}$ in the system. Because the resolution is set to be $n_x = 128$ for $\tilde{T}$ and $\tilde{q}$, the maximum number of allowable Fourier modes in the system is 64. We scan the $k_{\rm max}$ in Eq. (3) from 8 to 64 by generating separated data sets, and training the neural network with $n_{\rm sample} = 10\,000$. The sudden dropping of MAE for all cases demonstrates that there is a threshold of minimum number of neurons required in each hidden layer $n_{\rm neuron}$ in order to accurately recover the Hammett–Perkins closure. The threshold is approximately

$$n_{\rm neuron}^{\rm th} = 2k_{\rm max}. \tag{5}$$

When $n_{\rm neuron} < n_{\rm neuron}^{\rm th}$, the testing MAE is always larger than 0.01; while $n_{\rm neuron} \geq n_{\rm neuron}^{\rm th}$, the testing MAE rapidly reduces an order of magnitude or more. The threshold $2k_{\rm max}$ coincides with the degrees of freedom of the input $\tilde{T}(x)$ in Fourier space representation—that is, amplitude A_k and phase shift ϕ_k are randomly chosen for each k, suggesting that MLP is able to learn the Fourier space information despite the fact that the training data are being fed into the configuration space. This also implies that if the input data contains full-spectrum information, then $n_{\rm neuron}$ shall be at least equal to n_x in order to resolve all Fourier modes. It should also be noted that based on our tests with a fixed number of training samples $n_{\rm sample} = 10\,000$, increasing $n_{\rm neuron}$ well beyond $n_{\rm neuron}^{\rm th}$ does not necessary improve the model performance dramatically especially for the low $k_{\rm max}$ cases. As for the depth of a model, no significant improvement in MAE was found as the number of hidden layers was gradually increasing from 1 to 6 ($\epsilon = 8.89 \times 10^{-4}$ to $\epsilon = 1.15 \times 10^{-3}$). However, reducing the number of layers may reduce the computational cost. Therefore, we conclude (1) the minimum number of neurons of hidden layers depends on the input training data—in general, $n_{\rm neuron}$ should be a little bit larger than $n_{\rm neuron}^{\rm th} = 2k_{\rm max}$, where $k_{\rm max}$ is the maximum wave number in the input data, and (2) a two-hidden-layer MLP is sufficient to accurately calculate the heat flux $\tilde{q}_i(x)$ for $\tilde{T}_i(x)$ given $n_{\rm neuron} \geq n_{\rm neuron}^{\rm th}$. Unless noted otherwise, the results in the rest of the paper were obtained with the training data set with $k_{\rm max} = 8$ and if applicable, a two-hidden-layer MLP with $n_{\rm neuron} = 64$.

We also train one classical deep learning model—the convolutional neural network (CNN). Here, we pick a widely used CNN architecture in computer vision studies. It consists of an input and an output layer, as well as multiple convolution layers, pooling layers, and up-sampling layers as hidden layers. The convolution layers and pooling layers generate a latent space representation for the input temperature profile, and the convolution layers and up-sampling layers decode the latent space representation to the output flux profile. The output of the convolution layer is the sliding dot product of its input and its *filters* where the *filters* detect the specific type of local features of the input. The weights of the filter are shared in the whole domain, which is reasonable assuming spatial symmetry. The pooling layer down-samples the input by combining multiple neurons in the previous layer into one neuron in the next layer, while the up-sampling layer duplicates one neuron in the previous layer to multiple neurons in the next layer. As shown in Fig. 4, the CNN appears unable to perfectly overfit the training data with $n_{\rm sample} = 10$, but eventually has a similar performance as MLP when the training and testing MAE converged at $n_{\rm sample} = 100\,000$. This shows that the CNN is more data hungry because it estimates multiple learnable parameters and requires more sample data to converge.

Due to the nature of the closure problem, the choice of activation function of the output layer is essential to getting correctly trained MLP and CNN. Because the heat flux profile has both positive and negative values, ReLU is no longer suitable as the activation function of the last layer since it only outputs in $[0, \infty)$. Figure 6 shows an example that the CNN fails to predict the output $\tilde{q}$ when ReLU is chosen as the activation function of the output layer. An appropriate activation function of the last layer is the linear function. When the activation function is switched from ReLU to linear, the CNN correctly predicts the output $\tilde{q}$ from the input $\tilde{T}$.

The last type of neural network we train is a two-layer discrete Fourier transform (DFT) neural network. DFT is different than MLP and CNN in the sense that it is intentionally designed to solve the Fourier transform type problem. It has no hidden layer—only the input and output layers; the activation function is linear for all neurons; and it trains the transformation matrix $\mathcal{W}$. For instance, in DFT, the relation between the input $\tilde{T}(x)$ and output $\tilde{q}(x)$ is $\tilde{q} = \mathcal{W}\tilde{T}$. Recall the definition of forward and inverse discrete Fourier transform

$$X_k = \mathcal{F}x_n, \quad x_n = \mathcal{F}^{-1}X_k, \tag{6}$$

where $x = \{x_n\}$ and $X = \{X_k\}$ are the signals in configuration and Fourier space, and $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the forward and inverse Fourier transfer matrices. The idea behind the DFT neural network is for the training weight matrix $\mathcal{W}$ to be close to the analytical form of the forward transform matrix $\mathcal{F} = \{\mathcal{F}_{jk}\} = \{\omega^{jk}/\sqrt{N}\}$ or inverse transform matrix $\mathcal{F}^{-1} = \{\mathcal{F}^{-1}_{jk}\}$, where N is the total sample in one period (i.e., n_x in our setup). Back to Hammett–Perkins closure Eq. (2), it can be rewritten in the DFT matrix form

$$\tilde{q}(x) = \mathcal{F}^{-1}\mathcal{K}\mathcal{F}\tilde{T}(x), \tag{7}$$

where $\mathcal{K} = \{k_{jk}\} = \{-i\text{sign}(k)\delta_{jk}\}$ is the matrix form of $-ik/|k|$. Therefore, training a DFT neural network to learn the Hammett–Perkins closure in configuration space is equivalent to learning the weight matrix $\mathcal{W} = \mathcal{F}^{-1}\mathcal{K}\mathcal{F}$. One would hence expect that DFT shall have a good performance due to the existence of an analytical expression [Eq. (7)] for the Hammett–Perkins closure.

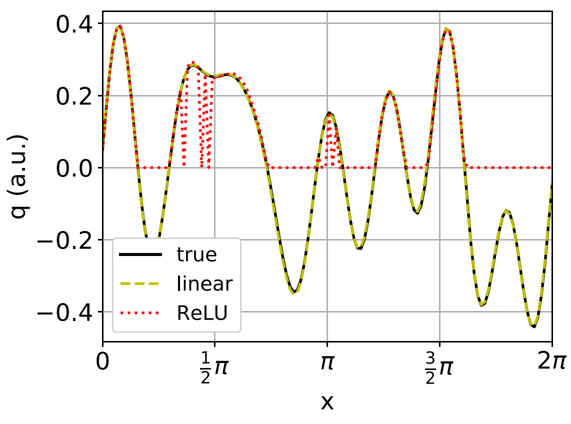

FIG. 6. Effect of activation function of the output layer. The black solid line represents the true value; the yellow dashed line represents the prediction with linear function; and the red dotted line represents the prediction with ReLU function.

The accuracy of the three different types of neural networks, indicated by the probability distribution function of the mean-absolute-error (MAE), is summarized in Fig. 7. All three models are trained with the same training data set with $n^{\text{train}}_{\text{sample}} = 1\,000\,000$, and tested with the same test data set with $n^{\text{test}}_{\text{sample}} = 10^4$. Note that the test data set is independent of the training data set. In this test, both MLP and CNN use ReLU as the activation functions for the hidden layers. The original two hidden layers MLP and CNN turn out to have a similar accuracy with CNN being slightly better than MLP ($\epsilon_{MLP} = 9.72 \times 10^{-4} \gtrsim \epsilon_{CNN} = 7.45 \times 10^{-4}$); while DFT is 4 orders more accurate than MLP and CNN ($\epsilon_{DFT} = 2.69 \times 10^{-8}$). The optimized MLP model by the Bayesian approach gives a more accurate prediction with $\epsilon^{\text{Bayesian}}_{MAE} = 5.49 \times 10^{-4}$ and its error distribution has a significant narrower ($\sim$50%) width compared with the original MLP error distribution. The impact of 10^{-3} level error from the neural network on its applications, e.g., whether the error accumulates in a long-term fluid simulation, is unclear at this point and is the subject of further studies.

Although both MLP and CNN are trained with mixed multi-mode data, they are able to resolve individual modes as well. Figure 8 shows how the MAE and standard deviation (STD) of the prediction errors change for single k testing data. The prediction errors of MLP (red line), CNN (green line), and optimized MLP (purple line) do not change significantly with the mode number k, suggesting all modes in the system are resolved equally well. It should be noted that CNN appears to be able to handle single mode cases exceptionally well—for MLP, the STD of single mode data has the same order as the STD of multiple mode data; for CNN, however, the STD of single mode data is an order smaller than the STD of multiple mode data. In multiple mode cases, MLP and CNN have similar deviations in the prediction error (Fig. 7).

A successful machine learning model should also be robust with respect to the uncertainty (or noise) presented in the simulation and experimental data due to truncation errors, random/stochastic processes, and/or the precision and accuracy of the measuring instrument. We therefore study the robustness of trained models by adding white noise $A_{\text{noise}}\tilde{T}_{\text{noise}}$ to the test data set $\tilde{T}$ and monitoring the model prediction $\tilde{q}_{\text{signal+noise}}$. Here, A_{noise} is the maximum amplitude of white noise, and the error metric MAE is evaluated by comparing $\tilde{q}_{\text{signal+noise}}$ to the true $\tilde{q}$ with clean $\tilde{T}$. Figure 9 shows how MAE varies with A_{noise} for three different models. The change of MAE for MLP and CNN has two phases: (1) when A_{noise} is smaller than the prediction error of the neural network (roughly ϵ), the MAE does not increase with A_{noise} as ϵ over-dominates A_{noise} and (2) when A_{noise} is larger than ϵ, the MAE starts to increase linearly with the noise amplitude A_{noise}. It also appears that MLP with two hidden layers is more resilient to noise, most likely because the trained network filters out the high-k noise with its multiple layers. We remark that although the Bayesian optimized model performs better than the two hidden layers MLP for the clean data, it gives less accurate predictions for the noisy data when $A_{\text{noise}} > 2 \times 10^{-3}$. In contrast, DFT seems to capture all levels of white noise because DFT essentially mimics the analytic expression of Hammett-Perkins closure and cannot "filter" any noise. It thus becomes less accurate than MLP or CNN when $A_{\text{noise}} > 2 \times 10^{-3} \sim \epsilon_{MLP,CNN}$. Overall, MLP is the most robust model and DFT is the least robust model in the noise test. The noise level from the PIC simulation is on the order of $1/\sqrt{n}$, where n is the number of particles per cell. Typically, $n = 50$ in the nonlinear PIC simulation; therefore, the

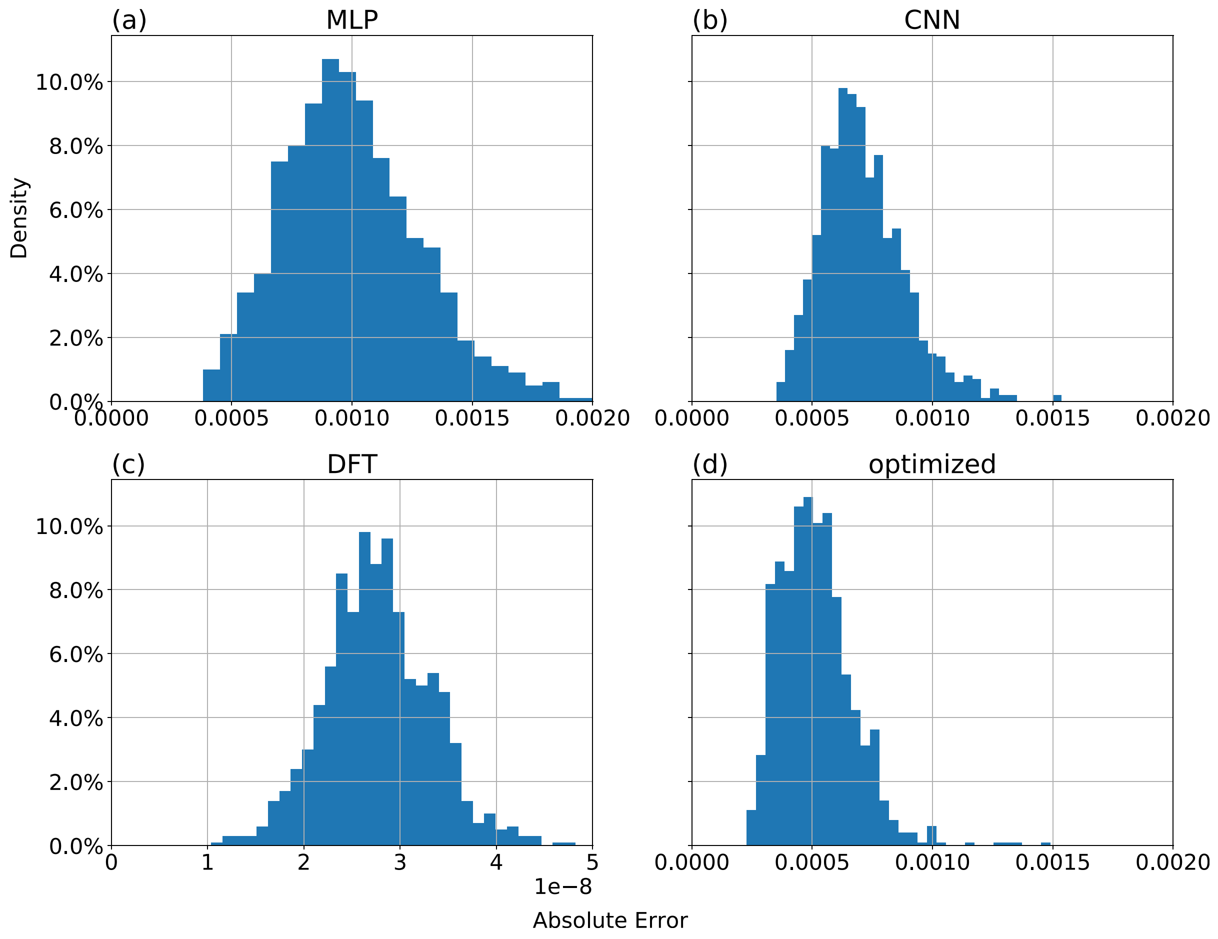

FIG. 7. Probability distribution function (PDF) of absolute error ϵ for different neural network models when $n_{\text{sample}} = 10^6$. (a) for MLP, (b) for CNN, (c) for DFT, and (d) for the Bayesian optimized model. The mean absolute errors for different NN are $\epsilon_{MLP} = 9.72 \times 10^{-4}$, $\epsilon_{CNN} = 7.45 \times 10^{-4}$, and $\epsilon_{DFT} = 2.69 \times 10^{-8}$.

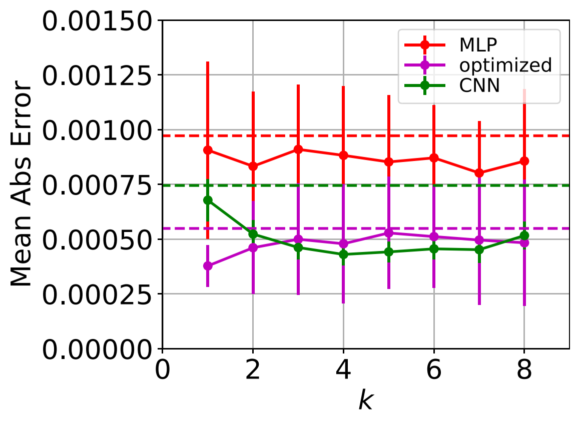

FIG. 8. The MAE (dots) and STD (bars) of the prediction errors of MLP (red), CNN (green), and Bayesian optimized MLP (purple) if the testing data only contains a single k perturbation.

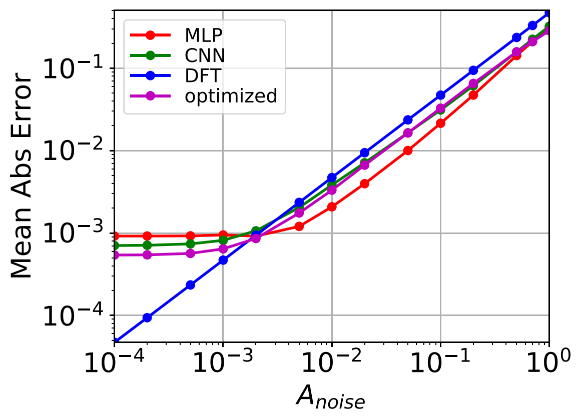

FIG. 9. The MAE with the amplitude of noise. Red line for MLP, green line for CNN, blue line for DFT, and purple line for the Bayesian optimized model.

noise level is around 14%. The two-hidden-layer MLP is the most accurate model among the ML surrogate models we studied in this paper.

V. SUMMARY AND OUTLOOK

Even though MLP and CNN are similar in performance for the 1D closure problem, we suspect that CNN will perform much better than MLP for future 2D and 3D closure problems as the convolution layer is even more efficient at handling 2D and 3D data.[23] In principle, the fully connected MLP can also be used to learn the features from 2D and 3D data. However, it becomes less practical as the number of neurons and weights greatly increase with the data dimension. For example, the number of weights for 3D data is $n_x n_y n_z$ for each neuron in a fully connected layer. On the other hand, the number of weights of a filter in the convolution layer is $n_{f,x} n_{f,y} n_{f,z}$, where $n_{f,i} < n_i (i = x, y, z)$ is the filter size in each direction, independent of the input size. In our current 1D study with 128 inputs/outputs, the evaluation cost for MLP is $11 \mu s$ per sample, for DFT it is $10 \mu s$ per sample, and for CNN it is $115 \mu s$ per sample. Although the evaluation cost for CNN is around an order of magnitude larger than those for MLP and DFT for this 1D case, it scales quadratically for MLP and DFT as $O(n^2)$ while for CNN it scales linearly as $O(n_k n)$, where n is the problem size and $n_k = 15$ is the kernel size. Therefore, for a 2D problem with 128×128 inputs/outputs, the evaluation costs for MLP, DFT, and CNN will approximately be 180 ms, 164 ms, and 14 ms, respectively. It is not feasible to use MLP or DFT in large scale 2D and 3D problems.

Generally, the fully connected layer performs better than the convolution layer for the 1D problem, such as the 1D Hammett–Perkins closure we discuss here. As for DFT, despite the fact that it has superior accuracy compared to the MLP and CNN, it may lose its advantage when training advanced closures with complicated, if not impossible, analytical forms, e.g., data from the fully nonlinear simulation of Vlasov-Poisson equations. The performance of MLP, CNN, and DFT for 2D/3D closure problems is beyond the scope of this study and is left to be explored in the future.

We emphasize that our work here is a proof-of-principle study to demonstrate that appropriate neural networks are capable of calculating the Landau fluid closure. It is our hope that the success of the training machine-learned Hammett–Perkins closure would help researchers to gain confidence in the machine/deep learning approach to the closure problem and also guide our future attempts on training more sophisticated closure models. Implementation of such a neural network model in global 3D fluid turbulence codes, for example, BOUT++[17] and GDB,[18] is on another level of complexity (e.g., nonuniform grid, non-periodic boundary conditions, and dependence on other plasma quantities). Further benchmarking the neural network based Landau fluid code with the existing Landau-fluid[26] and Gyro-Landau-fluid module[27,28] of BOUT++ is an on-going research and will be reported in future publications.

ACKNOWLEDGMENTS

The authors would like to thank Dr. X. Z. Tang, Dr. R. Maulik, Dr. C. K. Huang, Dr. L. Wang, and Dr. C. F. Dong for the helpful discussions. C. H. Ma and W. X. Wang are supported by DOE Contract No. DE-AC02-09CH11466 from the Princeton Plasma Physics Laboratory. B. Zhu and X. Q. Xu are supported by DOE Contract No. DE-AC52-07NA27344 from the Lawrence Livermore National Laboratory. C. H. Ma, W. X. Wang, B. Zhu, and X. Q. Xu are supported by SciDAC Tokamak Disruption Simulation project. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Lawrence Livermore National Laboratory IM No. LLNL-JRNL-791193.

REFERENCES

[1] L. Spitzer, Jr. and R. Härm, Phys. Rev. **89**, 977 (1953).
[2] S. Braginskii, Rev. Plasma Phys. **1**, 205 (1965).
[3] G. W. Hammett and F. W. Perkins, Phys. Rev. Lett. **64**, 3019 (1990).
[4] Z. Chang and J. Callen, Phys. Fluids B **4**, 1167 (1992).
[5] M. Umansky, A. Dimits, I. Joseph, J. Omotani, and T. Rognlien, J. Nucl. Mater. **463**, 506 (2015).
[6] Z. Guo and X.-Z. Tang, Phys. Rev. Lett. **108**, 165005 (2012).
[7] P. Hunana, G. Zank, M. Laurenza, A. Tenerani, G. Webb, M. Goldstein, M. Velli, and L. Adhikari, Phys. Rev. Lett. **121**, 135101 (2018).
[8] L. Wang, B. Zhu, X-q Xu, and B. Li, AIP Adv. **9**, 015217 (2019).
[9] B. K. Spears, J. Brase, P.-T. Bremer, B. Chen, J. Field, J. Gaffney, M. Kruse, S. Langer, K. Lewis, R. Nora *et al.*, Phys. Plasmas **25**, 080901 (2018).
[10] D. R. Ferreira, and J. Contributors, preprint arXiv:1811.00333 (2018).
[11] M. Boyer, S. Kaye, and K. Erickson, Nucl. Fusion **59**, 056008 (2019).
[12] B. Cannas, A. Fanni, P. Sonato, and M. K. Zedda, and J.-E. Contributors, Nucl. Fusion **47**, 1559 (2007).
[13] C. Rea and R. S. Granetz, Fusion Sci. Technol. **74**, 89 (2018).
[14] J. Kates-Harbeck, A. Svyatkovskiy, and W. Tang, Nature **568**, 526 (2019).
[15] O. Meneghini, S. P. Smith, P. B. Snyder, G. M. Staebler, J. Candy, E. Belli, L. Lao, M. Kostuk, T. Luce, T. Luda *et al.*, Nucl. Fusion **57**, 086034 (2017).
[16] J. Peterson, K. Humbird, J. Field, S. Brandon, S. Langer, R. Nora, B. Spears, and P. Springer, Phys. Plasmas **24**, 032702 (2017).
[17] B. Dudson, M. Umansky, X. Xu, P. Snyder, and H. Wilson, Comput. Phys. Commun. **180**, 1467 (2009).
[18] B. Zhu, M. Francisquez, and B. N. Rogers, Comput. Phys. Commun. **232**, 46 (2018).
[19] A. Dimits, I. Joseph, and M. Umansky, Phys. Plasmas **21**, 055907 (2014).
[20] F. Chollet, https://keras.io for "Keras," (2015).
[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, tensorflow.org for "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," (2015).
[22] S. S. Haykin, *Neural Networks and Learning Machines/Simon Haykin* (Prentice Hall, New York, 2009).
[23] Y. LeCun, Y. Bengio, and G. Hinton, Nature **521**, 436 (2015).
[24] R. Velik, in *International Conference on Computational Intelligence and Security* (IEEE, 2008), Vol. 1, pp. 120–123.
[25] D. P. Kingma and J. Ba, preprint arXiv:1412.6980 (2014).
[26] C. Ma, X. Xu, P. Xi, and T. Xia, Phys. Plasmas **22**, 010702 (2015).
[27] C. Ma, X. Xu, P. Xi, T. Xia, P. Snyder, and S. Kim, Phys. Plasmas **22**, 055903 (2015).
[28] C. Ma and X. Xu, Nucl. Fusion **57**, 016002 (2017).