Building high accuracy emulators for scientific simulations with deep neural architecture search

IOP Publishing Mach. Learn.: Sci. Technol. 3 (2022) 015013 https://doi.org/10.1088/2632-2153/ac3ffa

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
21 April 2021

REVISED
9 September 2021

ACCEPTED FOR PUBLICATION
3 December 2021

PUBLISHED
27 December 2021

PAPER

Building high accuracy emulators for scientific simulations with deep neural architecture search

M F Kasim[1,*], D Watson-Parris[2], L Deaconu[2], S Oliver[3], P Hatfield[1], D H Froula[4], G Gregori[1], M Jarvis[5], S Khatiwala[3], J Korenaga[6], J Topp-Mugglestone[1], E Viezzer[7,8] and S M Vinko[1]

[1] Clarendon Laboratory, Department of Physics, University of Oxford, Parks Road, Oxford, United Kingdom
[2] Atmospheric, Oceanic and Planetary Physics, Department of Physics, University of Oxford, Oxford, United Kingdom
[3] Department of Earth Sciences, University of Oxford, Oxford, United Kingdom
[4] Laboratory for Laser Energetics, University of Rochester, Rochester, NY, United States of America
[5] Denys Wilkinson Building, Department of Physics, University of Oxford, Keble Road, Oxford, United Kingdom
[6] Department of Geology and Geophysics, Yale University, New Haven, CT, United States of America
[7] Department of Atomic, Molecular and Nuclear Physics, University of Seville, 41012 Seville, Spain
[8] Max-Planck-Institut für Plasmaphysik, EURATOM Association, Boltzmannstr. 2, 85748 Garching, Germany
[*] Author to whom any correspondence should be addressed.

E-mail: muhammad.kasim@physics.ox.ac.uk

Keywords: emulators, simulations, neural architecture search, deep learning

Supplementary material for this article is available online

Abstract

Computer simulations are invaluable tools for scientific discovery. However, accurate simulations are often slow to execute, which limits their applicability to extensive parameter exploration, large-scale data analysis, and uncertainty quantification. A promising route to accelerate simulations by building fast emulators with machine learning requires large training datasets, which can be prohibitively expensive to obtain with slow simulations. Here we present a method based on neural architecture search to build accurate emulators even with a limited number of training data. The method successfully emulates simulations in 10 scientific cases including astrophysics, climate science, biogeochemistry, high energy density physics, fusion energy, and seismology, using the same super-architecture, algorithm, and hyperparameters. Our approach also inherently provides emulator uncertainty estimation, adding further confidence in their use. We anticipate this work will accelerate research involving expensive simulations, allow more extensive parameters exploration, and enable new, previously unfeasible computational discovery.

1. Introduction

Finding a general approach to speed up a large class of simulations would enable tasks that are otherwise prohibitively expensive and accelerate scientific research. For example, fast and accurate simulations promise to speed up new materials and drug discovery [1] by allowing rapid screening and ideas testing. Accelerated simulations also open up novel possibilities for online diagnostics for cases like x-ray scattering in plasma physics experiments [2] and to monitor edge-localized modes in magnetic confinement fusion [3], enabling real-time prediction-based experimental control and optimization. However, for such applications to be successful the simulations need not only be fast but also accurate; achieving both to the level required for advanced applications remains an active objective of current research.

One popular approach to speeding up simulations is to train machine learning models to emulate slow simulations [4–7] and use the emulators instead. The main challenge in constructing emulators with machine learning models is in their need for large amounts of training data to achieve the required accuracy in replicating the outputs of the simulations. This training data could be prohibitively expensive to generate with slow simulations.

To construct high fidelity emulators with limited training data, the machine learning models need to have a good prior on the simulation models. Most work to date in building emulators, using random

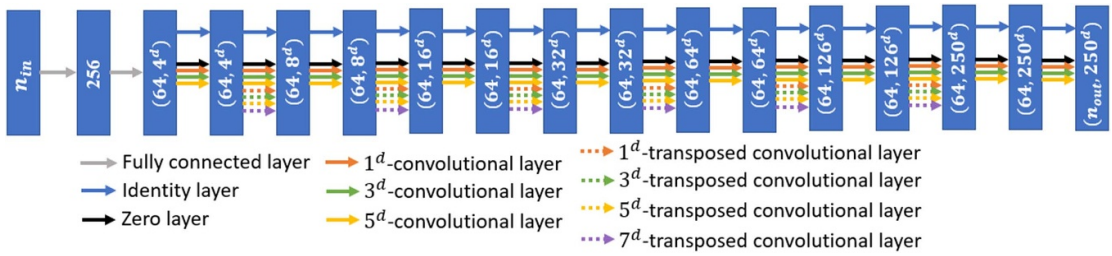

Figure 1. The super-architecture used in this paper where d is the dimension of the output signal. The first numbers in the brackets indicate the number of channels and the last numbers indicate the signal size. For cases where the output signal are not 250^d, then all the sizes in the intermediate nodes are scaled accordingly. Close arrows indicate the operations in the same group. The output of the identity layer and the selected of convolutional layer are added into the destination node.

forests [4], Gaussian Processes [5], or other machine learning models [6, 7], do not fully capture the correlation among the output points, limiting their accuracy in emulating simulations with one, two, or three-dimensional output signals. On the other hand, convolutional neural networks (CNN) have shown to have a good prior on natural signals [8], making them suitable for processing natural n-dimensional signals. However, as the CNN priors inherently rely on their architectures [8], one has to find an architecture that gives the suitable prior for a given problem. Manually searching for the right architecture can take a significant amount of time and domain-specific expertise, and often produces sub-optimal results.

Here we propose to address this problem by employing efficient neural architecture search [9, 10] to simultaneously find the neural network architecture that is well-suited for a given case and train it. With the efficient neural architecture search and a novel super-architecture presented in this work, the algorithm can find and train fast emulators for a wide range of applications while offering major improvements in terms of accuracy compared with other techniques, even when the training data is limited. We call the presented method Deep Emulator Network SEarch (DENSE).

DENSE is largely inspired by ProxylessNAS [10] which starts by defining the search space of neural network architectures in a form of super-architecture. A super-architecture consists of multiple nodes where the first node represents the simulation inputs and the last node the predicted simulation outputs. Each pair of nodes is connected by multiple groups of operations. Each group consists of a set of operations, such as 1×1 convolution, 3×3 convolution, or similar. Most of the operations, such as convolution, contain sets of trainable values that are commonly known as *weights*. In one forward calculation of the neural network (i.e. predicting a set of outputs given some input), only one operation per group is chosen according to its assigned probability. The probability of an operation being chosen is determined by a trainable value associated with the operation, which we call the *network variable*.

The super-architecture used in this work is shown in figure 1. In every group in the super-architecture, there are convolutional layers with different kernel sizes and a zero layer that multiplies the input with zero. The option of having a zero layer and multiple convolutional layers enables the algorithm to choose the appropriate architecture complexity for a given problem. The super-architecture also contains skip connections [11] (i.e. identity layers) to make it easier to train.

The super-architecture in figure 1 can be adapted to different output dimensions, including scalar outputs (0D), 1D function outputs, 2D images, or 3D profiles, subject to the memory availability. The adaptation can be done by choosing the appropriate convolution and transposed convolution operators. For example, if the outputs are 2D images, then 2D convolutions are chosen.

Test cases with 0D outputs are special in that the convolution operators become standard linear operators—matrix multiplication. Although the difference between various sizes of convolutional layers is no longer present, the zero layers in the super-architecture can still provide options for the super-architecture to choose lower-complexity architectures.

Training the neural network involves two update steps. In the first step, an operation for each group is chosen according to its probability, forming one possible architecture. The weights, $\mathbf{w}$, of the selected operations are then updated to minimize the expected value of a defined loss function, $\mathcal{L}$, between the predicted simulation output and the actual simulation output,

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha_1 \nabla_{\mathbf{w}} \mathbb{E}_{a\sim\mathcal{A}(\mathbf{b})} \left[\mathcal{L}(\mathbf{w}|\mathbf{X_t}, \mathbf{y_t}, a)\right], \tag{1}$$

where α_1 is the update step size, $\mathbf{X_t}$ and $\mathbf{y_t}$ are the input and output from the training dataset, a is an architecture sampled from the super-architecture $\mathcal{A}(\mathbf{b})$ according to the network variables, $\mathbf{b}$. Randomly choosing an operation in each group acts as a regularization during the training process, which can minimize

Table 1. Summary of test cases considered in this paper. The inputs to all simulations are all 0D (scalars). The number of datasets for relatively fast simulations is capped to 14 000 for convenience. For slower simulations the size of the dataset is limited by time and resource constraints.

No.	Test case	# Inputs	# Outputs	Output type	# Dataset	Avg. simulation running time
1	XRTS	3	1	1D (250 points)	14 000	15 s
2	OTS	5	1	1D (250 points)	14 000	15 s
3	XES	10	1	1D (250 points)	14 000	20 min
4	ELMs	14	10	1D (250 points)	14 000	15 min
5	Halo	5	1	1D (250 points)	14 000	5 s
6	ICF JAG	5	4	2D (64 × 64)	10 000	30 s
7	ICF JAG Scalars	5	15	0D (scalar)	10 000	30 s
8	SeisTomo	13	1	1D (250 points)	6100	2 h
9	MOPS	6	45	2D (128 × 64)	410	144 CPU-hours
10	GCM	3	12	2D (192 × 96)	39	1150 CPU-hours

overfitting. The loss function in this paper is defined as the Huber loss function [12] to minimize the effect of outlier data and increase robustness.

The second update step involves evaluating the performance of various sampled architectures on the validation dataset, which is different from the training dataset employed in the first step. The performance of an architecture can be evaluated based on the loss function, inference time, power consumption, or some other combination of relevant criteria. The architectures are then ranked based on their performance and they are given rewards according to their rank. The network variables, $\mathbf{b}$, are updated to increase the probability of the high-ranked architectures and decrease the probability of the low-ranked architectures. Formally, the update can be written as [13],

$$\mathbf{b} \leftarrow \mathbf{b} + \alpha_2 \, \mathbb{E}_{a\sim\mathcal{A}(\mathbf{b})} \left(\mathcal{R}_a \nabla_{\mathbf{b}} \log \left[\pi(a|\mathbf{b}) \right] \right), \tag{2}$$

where α_2 is the update step size, $\mathcal{R}_a$ is the reward value given to the architecture a based on its rank, and the function $\pi(a|\mathbf{b})$ is the likelihood of the architecture a being chosen given the network variables $\mathbf{b}$.

In this case, we ranked the architectures based on the Huber loss [12] on the validation dataset and gave the rewards to follow the zero-mean ranking function in CMA-ES [14]. The use of a zero-mean ranking function reduces the update variance and makes the update step scale-invariant, increasing the robustness of the algorithm.

2. Results

The combined update steps from equations (1) and (2), and the use of a ranking function in assigning rewards, make DENSE a robust algorithm to simultaneously learn the weights and find the right architecture for a given problem. To illustrate this, we apply the method to ten distinct scientific simulation cases: inelastic x-ray Thomson scattering (XRTS) in high-energy-density physics [2, 15], optical Thomson scattering (OTS) in laboratory astrophysics [16], tokamak edge-localised modes diagnostics (ELMs) in fusion energy science [3], x-ray emission spectroscopy (XES) in plasmas [17, 18], galaxy halo occupation distribution modelling (Halo) in astrophysics [19], seismic tomography of the Shatsky Rise oceanic plateau (SeisTomo) [20], global aerosol-climate modelling using a general circulation model (GCM) in climate science [21], oceanic pelagic stoichiometry modelling (MOPS) in biogeochemistry [22], and neutron imaging (ICF JAG) and scalar measurements (ICF JAG Scalars) in inertial confinement fusion experiments [23].

The tested simulations have ranging numbers of scalar input parameters from 3 to 14, and span outputs from 0D (scalars) to multiple 2D signals (images). Datasets for simulations that run in less than 1 CPU-hour were generated by running them 14 000 times with random sets of inputs. For more expensive simulations, the number of generated dataset is limited by the time and resources available (see table 1 for more detailed information). Each dataset is divided into three parts: 50% is used as the training dataset, 21% for validation, and 29% as the test dataset. The test dataset was used only to present the results in this paper, never to build the models. The hyperparameters (including the depth and width of the super-architecture) were obtained by optimizing the result for OTS with CMA-ES [14, 24], and then used for the other cases without further tuning.

2.1. Emulator results

The example outputs of the trained emulators with DENSE for some cases are shown in figure 2. The outputs for other cases can be seen in figure 7. We see that the output of the emulators generally matches closely the

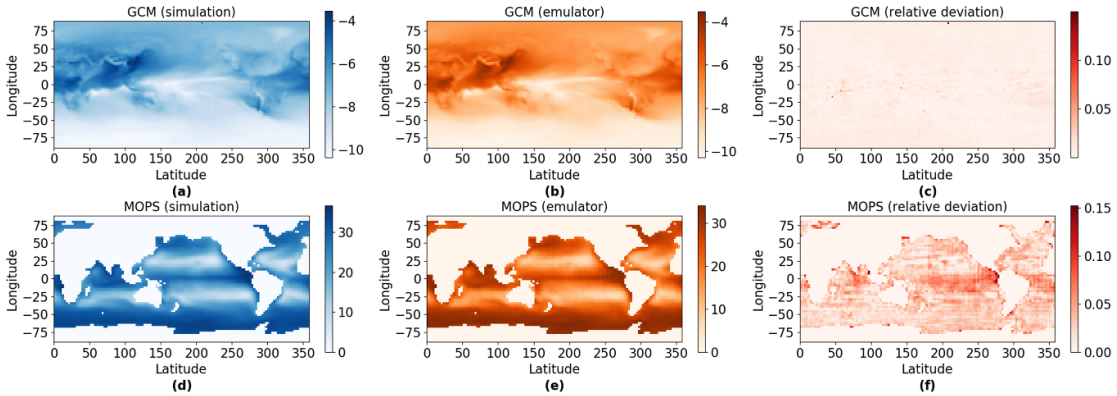

Figure 2. Original simulations (a) and (d) compared with outputs from their DENSE emulators (b) and (e) on a representative example from the test dataset. The relative differences are shown in (c) and (f). Outputs from the remaining eight cases are given in the methods section in figure 7.

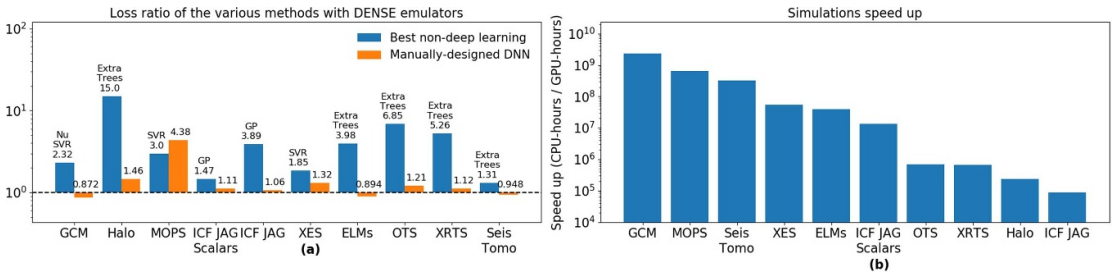

Figure 3. (a) The ratio between the loss function obtained by DENSE emulators and the best loss function found by non-deep learning methods and a manually-designed deep neural network. (b) The achieved speed up of the emulators using a GPU compared with the original simulations.

output of the actual simulations, even in MOPS and GCM where only 410 and 39 data points are available. When only a limited number of data is available, the choice of model architectures that give the right priors become important. Complex model architectures with bad priors could still fit the sampled training data, but are likely to overfit, giving bad accuracy on the out-of-samples data.

With DENSE, the model architecture that gives a good prior on the problem is automatically searched for by preferring models that can fit well the out-of-samples data (i.e. the validation dataset). Moreover, randomly choosing an operation in every layer acts as a regularizer in updating the weights during the training to minimize overfitting. These two advantages make DENSE suitable for learning to emulate a wide range of simulations including expensive ones where only a limited number of datasets can be generated.

While the simulations presented typically run in minutes to days, the DENSE emulators can process multiple sets of input parameters in milliseconds to a few seconds with one CPU core, or even faster when using a GPU card. For the GCM simulation which takes about 1150 CPU-hours to run, the emulator speedup is a factor of 110 million on a like-for-like basis, and over 2 billion with a GPU card. The speed up achieved by DENSE emulators for each test case is shown in figure 3(b).

Compared with other non-deep learning techniques usually employed in building emulators [25], the models found and trained by DENSE achieved the best results in all tested cases, and in most cases by a significant margin. The DENSE model also performs better than an emulator designed specifically for ICF JAG simulation [26] (i.e. CycleGAN in figure 1 of the supplementary material (available online at stacks.iop.org/MLST/3/015013/mmedia)). As seen in figure 3(a), the emulators built by DENSE achieved a loss function up to 14 times lower than the best performing non-deep learning model.

We also compared DENSE with a manually-designed deep neural network model by an architecture from the super-architecture in figure 1, where all the convolutional layers have size 3. The use of kernel size 3 and skip connections follows the idea of ResNet [11]. As with DENSE, the hyperparameters for manually-designed deep neural network were tuned to optimize its result for OTS.

Shown in figure 3(a) is the comparison between DENSE emulators and manually-designed deep neural network. Although in most cases their performances are similar, in some cases DENSE emulators can give a

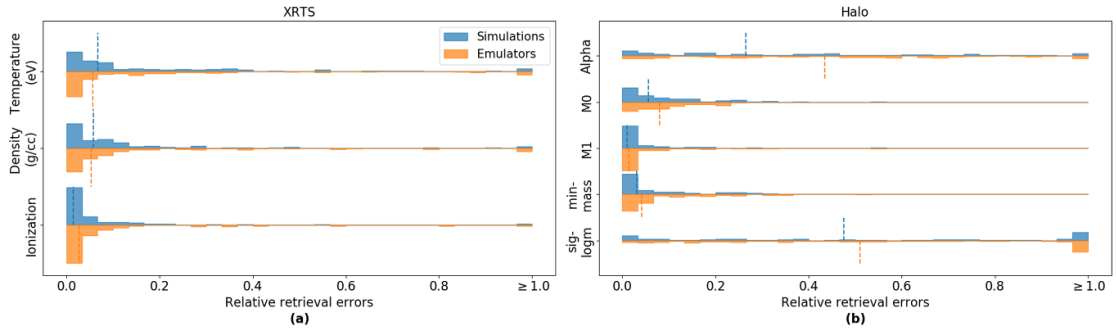

Figure 4. Histograms of the relative retrieval errors in solving the inverse problem with noisy data for the XRTS (a) and Halo (b) test cases. Retrieval conducted using the full simulations is shown in blue, and the retrieval using DENSE emulators in orange. Median values of the distributions are shown with the dashed lines.

considerable improvement in terms of loss function, as can be seen in Halo and MOPS. This illustrates the robustness of DENSE emulators in a wide range of cases.

2.2. Emulators for inverse problems

The high fidelity emulators built by DENSE are sufficiently accurate to allow us to substitute simulations even for more advanced tasks such as solving the inverse problem [27]. To illustrate this, we took a simulated output signal randomly from the test dataset where the actual parameters are known. A small noise ($\sim$1%) was added to the chosen signal to closely mimic a real observed signal from an experiment. Using this signal, we use an optimization algorithm [28] to retrieve the input parameters by minimizing the error between the sample signal and the output of the emulators, which we call it the *retrieval error*.

The results of the parameter retrieval using the emulators are compared with the retrieval using the simulations in figure 4, where we plot the relative retrieval error histograms for two cases. We observe that the relative retrieval errors from the emulators are very similar to those from the simulations.

Without much loss in accuracy, the parameter retrieval with the emulators only takes about 800 ms with a single GPU card. This is to be compared with using the actual simulations which could take up to 2 d (XES) even when using 32 CPU cores. As the parameters can be retrieved in less than one second rather than in hours or days, one can envisage employing this technique for online diagnostics, real-time data interpretation, or even on-the-fly intelligent automation with an accuracy comparable to high-fidelity simulations that are by far too computationally expensive to be used directly. The use of DENSE emulators also enables parameter retrieval with resource-intensive simulations, such as MOPS and GCM, that were too expensive before.

In addition to interpreting signals and parameter retrieval, the emulators can also be used to significantly speed up Bayesian uncertainty quantification [27]. Bayesian uncertainty quantification is usually done by constructing Bayesian posterior distribution using Markov Chain Monte Carlo (MCMC) algorithms. However, the cost of running MCMC to collect sufficient samples from the Bayesian posterior distribution is typically much larger than the cost for parameter retrieval, and is often intractable in practice. Here we perform the Bayesian posterior sampling using an ensemble MCMC algorithm [29] with the same conditions as in [27]. In short, we collect all parameters sets that produce spectra that lie in a certain band around a central spectrum.

Figure 5 compares the results of sampling the posterior distribution using simulations and emulators in two cases to interpret scattering and spectroscopy data. The posterior distribution sampled by the emulators are very similar to those by actual simulations, and we see that the emulators are well-suited to capture the correlations between parameters. However, note that while collecting 200 000 XES samples via simulations takes over 22 d, the sampling process with the emulators was completed in just a few seconds. Interestingly, building the emulator for XES from scratch only needs some 14 000 samples plus 8 h for training, so the whole pipeline to build the emulator and use it for MCMC is still considerably faster than directly collecting 200 000 samples using the original simulation.

2.3. Prediction uncertainty

A final important advantage of building emulators with DENSE is the availability of an intrinsic estimator of the emulator's prediction uncertainty. The randomization of network architectures from the super-architecture can be seen as a special case of dropout [30]. Thus, by adapting the theory of prediction uncertainty with Monte Carlo (MC) dropout by [31], we can show that DENSE emulators can produce the

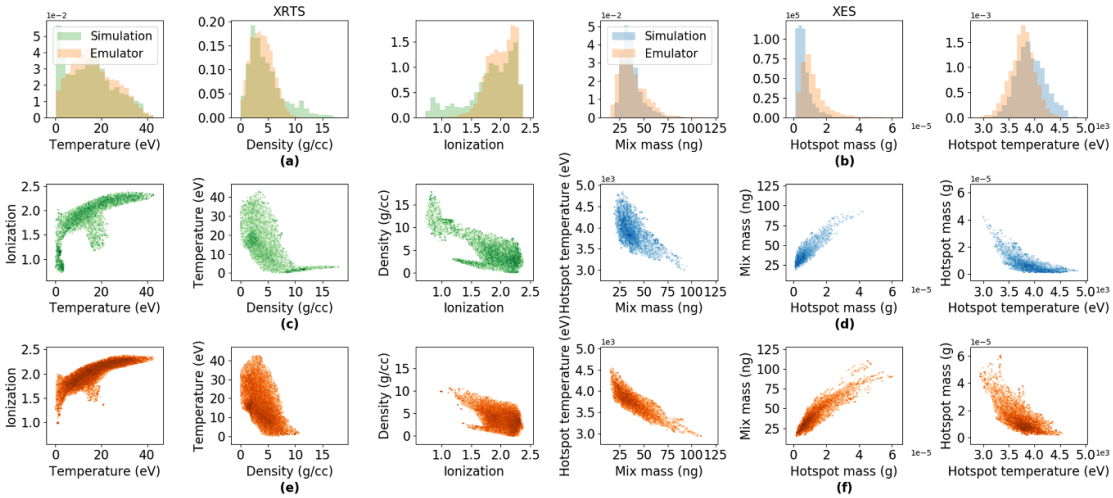

Figure 5. Comparison of results of Bayesian posterior sampling when applied using simulations and emulators in solving inverse problems. Histograms for the posterior distributions are shown for three retrieved parameters in the XES (a) and XRTS (b) cases. The parameter posterior distribution scatter plots are shown in panels (c), (d) for the simulations, and (e), (f) for the DENSE emulators.

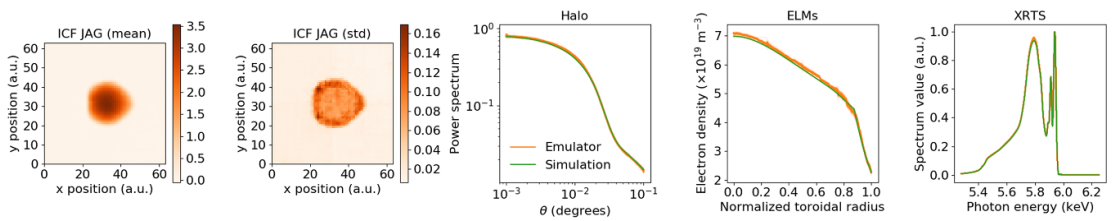

Figure 6. Emulator prediction uncertainties: comparison of DENSE emulator predictions with outputs from simulations for 2D (ICF JAG) and 1D models (Halo, ELMs, and XRTS). The orange lines in Halo, ELMs, and XRTS resemble the distribution of the emulators outputs while the green lines are the output from the simulations.

uncertainty of their outputs. The expected value and variance of a DENSE emulator prediction can be obtained by

$$\begin{aligned}\mathbb{E}(\mathbf{y}|\mathbf{x}) &= \mathbb{E}_{a\sim\mathcal{A}(\mathbf{b})}\left(\mathbf{y}|\mathbf{x},a\right)\\ \mathrm{Var}(\mathbf{y}|\mathbf{x}) &= \mathrm{Var}_{a\sim\mathcal{A}(\mathbf{b})}\left(\mathbf{y}|\mathbf{x},a\right),\end{aligned} \tag{3}$$

where a is the architecture sampled from the super-architecture $\mathcal{A}$ based on the final values of the network parameters, $\mathbf{b}$. Figure 6 shows the prediction uncertainty of the DENSE emulators for some cases, illustrating regions where they are either uncertain or confident in their predictions. The prediction uncertainty for other cases can be seen in figure 8.

As also observed in MC dropout [32], the prediction uncertainty in this case can be smaller than the difference between the predicted and simulated output, indicating an overconfident prediction. This problem of overconfidence can be resolved by stopping the training of network variables early. The investigation of prediction uncertainty tuning will be the subject of future work.

3. Discussions

3.1. Limitations

Although DENSE has the capability of emulating a wide range of simulations, it is still limited to simulations with a few scalar inputs. The DENSE algorithm has not been tested on building emulators with one, two, or three-dimensional direct inputs. One way to fit a simulation with multi-dimensional inputs to DENSE is by parameterizing the inputs using several scalar parameters (as done in ELMs) or employing a dimensionality reduction techniques [33].

Another limitation observed in DENSE is that it does not learn very well in regions where output variability is high, i.e. where changing the input parameters slightly changes the outputs significantly. This limitation is also observed in other cases of deep learning [34]. Due to the difficulty in learning, regions with

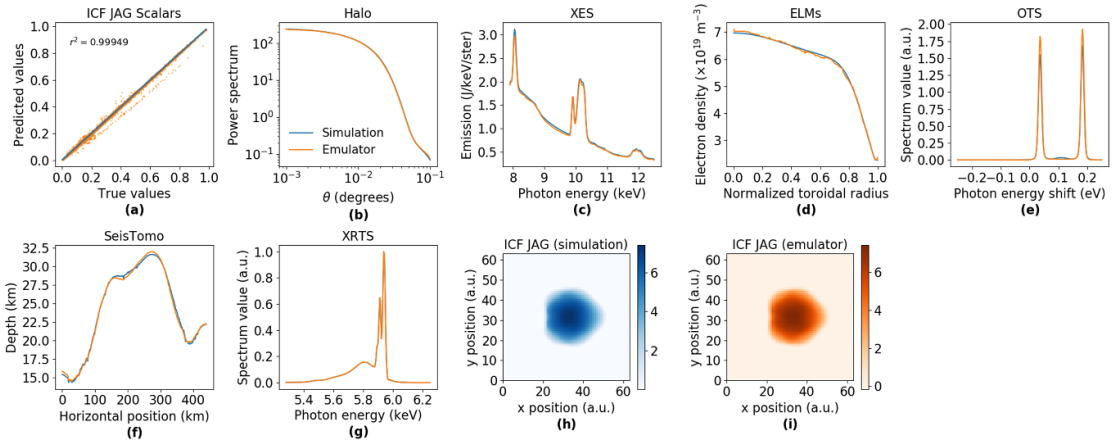

Figure 7. (a)–(i) Examples of emulator outputs compared with the simulation outputs for the remaining test cases not shown on figure 2 (i.e. ICF JAG, Halo, XES, ELMs, OTS, SeisTomo, and XRTS).

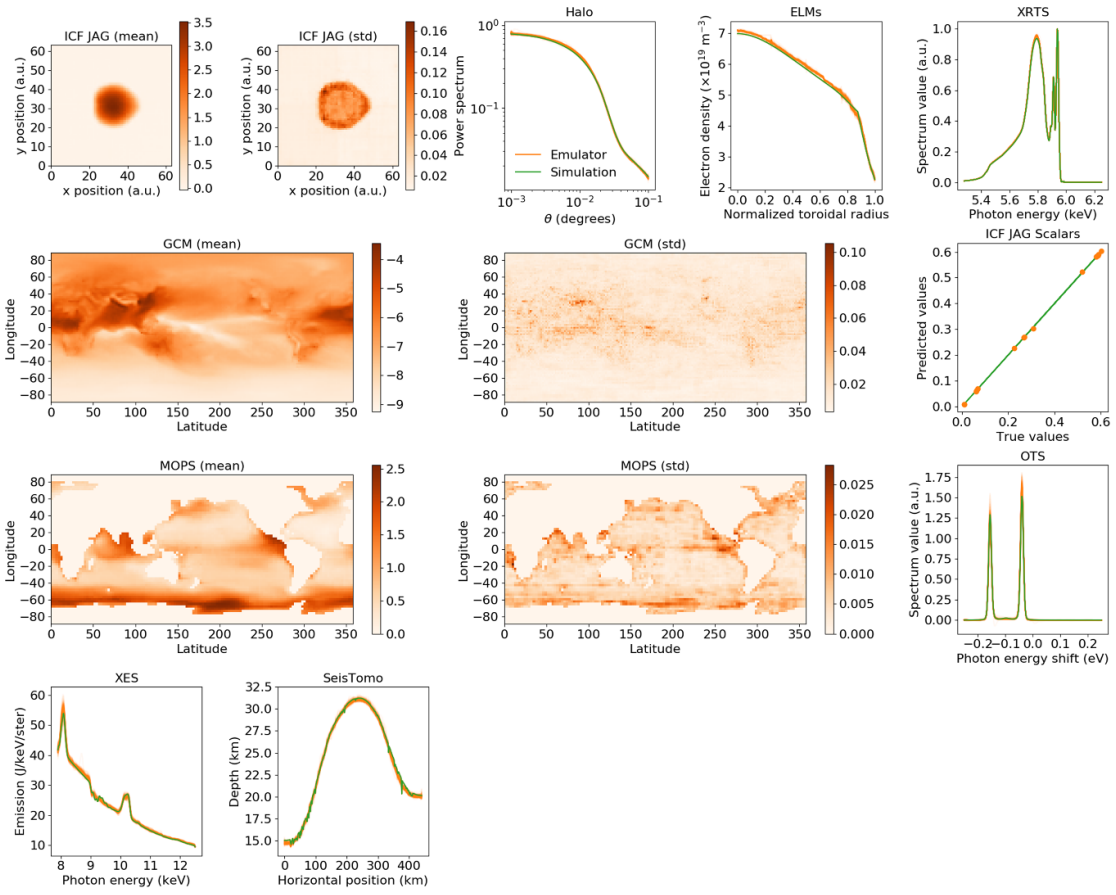

Figure 8. Examples of emulator uncertainties for test cases not shown in figure 6 (i.e. ICF JAG, Halo, XRTS, GCM, MOPS, OTS, XES, SeisTomo). The orange lines in 1D simulations resemble the distribution of the emulators outputs while the green lines are the outputs from the simulations.

high variability tend to require more samples than regions with low variability. This problem can thus be overcome by sampling the parameter space intelligently [35].

3.2. Applications

Our DENSE approach opens up numerous applications that require fast calculations. One of the main applications of DENSE emulators is real-time diagnostics of complex systems. For research in some fields, such as in plasma physics, diagnostics are usually done by solving an inverse problem using simulations, which involves running the simulations hundreds of times or more. By using the fast emulators instead of simulations, solving the inverse problem could be significantly accelerated without sacrificing the quality of

the solutions, as shown in section 2.2. Real-time diagnostics are the key in automating operations of some machines with complex systems, such as tokamaks [36] and particle accelerators [37].

Another application is the optimization of a very expensive simulations where the simulations can only be executed a few times. The emulators can be used to make high-quality guesses about the optimum parameters which can then be tested using the expensive simulations. This idea of utilizing a cheap model for expensive simulation optimization has been used in surrogate-model optimization [38] and Bayesian optimization [39]. By using a more accurate cheap model, such as DENSE emulators, the number of simulations to be executed can be much lower than in previous approaches. This is a potential avenue for future works.

4. Conclusions

We have shown that DENSE, a method based on neural architecture search, can be used to robustly build fast and accurate emulators for various types of scientific simulations even with limited training data. The capability of DENSE to accurately emulate simulations with limited data makes the acceleration of very expensive simulations possible. With the achieved acceleration of up to 2 billion times, DENSE emulators enable tasks that were impossible before, such as real-time simulation-based diagnostics, uncertainty quantification, and extensive parameters exploration. This large acceleration in solving inverse problems removes the barriers of using high fidelity simulations in real-time measurements, opening up new types of online diagnostics in the future. The wide range of successful test cases presented here shows the generality of the method in speeding up simulations, enabling rapid idea testing and accelerating new discovery across the sciences and engineering.

5. Methods

5.1. Test cases

Here we provide a description of the test cases employed in the paper. A summary of the test case parameters is given in table 1. All 1D simulation outputs are sampled to 250 points for convenience.

5.1.1. X-ray Thomson scattering (XRTS)

XRTS is a technique widely used in high-energy-density physics to extract plasma temperatures and densities by measuring the spectrum of an inelastically scattered x-ray pulse [2, 40]. The spectrum of the scattered light can be calculated from a set of plasma conditions and the scattering geometry [15]; this forms the simulation on which our emulator is based.

In this paper we consider the specific experimental case presented in [2] where three parameters (temperature, ionization, and density) are to be retrieved from a spectrum of x-rays scattered at a 90-degree angle from a shock-compressed Beryllium plasma. The high-speed emulator for XRTS enables fast solutions to the inverse problem and access to statistical information on the intrinsic uncertainty of the experiment, allowing better control of the experimental optimization and information extraction.

5.1.2. Optical Thomson scattering (OTS)

OTS is conceptually similar to XRTS except that it uses optical light instead of x-rays. Optical Thomson scattering is used in measuring electrons and ions temperatures and densities, as well as the flow speed of the plasma using the Doppler shift [16].

Here we considered retrieving five physical parameters (electron and ion temperatures, electron density, ionization, and flow speed) from a normalized scattered spectrum. The impact of building an emulator for OTS is similar to XRTS as it enables access to real-time data interpretation and to uncertainty quantification.

5.1.3. X-ray emission spectroscopy (XES)

X-ray emission spectroscopy is a general technique to probe a system by measuring the emitted spectrum and matching it with simulations or theoretical models. In this paper, we consider the diagnostic case of a laser-driven implosion experiment at the National Ignition Facility [17], using the spectroscopic model based on the CRETIN atomic kinetics code described in detail in [18].

5.1.4. Edge-localized modes (ELMs) diagnostics

ELMs are magnetohydrodynamic instabilities that occur in magnetically confined fusion plasmas with high confinement [41]. ELMs are explosive events and cause detrimental heat and particle loads on the plasma facing components of a tokamak. Various diagnostics are implemented to track ELMs [42]. Here, we compare the emulator to the predictive model [43] for the temporal evolution of the electron density profile using the transport code ASTRA [44].

The 14 input parameters in this case describe the diffusion, convective velocity, and particle source profiles [45] as a function of toroidal radial position and time. The output observable is the time-dependent electron density as a function of toroidal radial position.

5.1.5. Galaxy halo occupation distribution modelling (Halo)
Here we considered simulations of the angular-scale correlation of a galaxy population. The simulation software Halomod [46] was used to calculate the correlation function, angular scales, redshifts and the cosmological model as described in [19]. The input parameters used in this simulation follows the parameters described in [47]. Fast parameter retrieval is of particular interest here as often researchers are interested in extracting parameters for multiple different galaxy populations.

5.1.6. JAG model for inertial confinement fusion (ICF JAG)
JAG simulates the observables from an inertial confinement fusion experiment [23]. There are five input parameters in the case considered here. One simulation with five input parameters produces four two-dimensional images and 15 scalar values. Constructing fast and accurate emulators of the model allows for a more efficient exploration of the parameters space, and to obtain optimum sets of parameters more efficiently.

5.1.7. Shatsky Rise seismic tomography (SeisTomo)
The case considered here is the seismic tomographic inversion problem of the Shatsky Rise oceanic plateau [20]. Given the input parameters that describe the initial velocity profile and regularization in the optimization, the software solves for the velocity structure and the crustal thicknesses as a function of position in the Shatsky Rise that matches the seismic reflection data. Performing uncertainty quantification of the tomographic inversion would require the execution of the software hundreds of thousand times which is very expensive without a fast emulator model.

5.1.8. Global aerosol-climate modelling (GCM)
The model considered here is ECHAM-HAM [21] which calculates the distribution and evolution of both internally and externally mixed aerosol species in the atmosphere and their affect on both radiation and cloud processes. The model simulates the aerosol absorption optical depth as the observable for every month in a year. The model receives three input parameters which are (a) a scaling of the emissions flux of Black Carbon (BC; the main absorbing aerosol species), (b) a scaling on the removal rate of BC through wet deposition, and (c) a scaling of the imaginary refractive index of BC (which determines it is absorptivity) between 0.2 and 0.8. All of these factors contribute to the different absorption aerosol optical depths we emulate.

The cost of running the model for one year (including three months of spin-up) is about 1150 CPU-hours which is prohibitively expensive when generating thousands of training data points. However, we have shown that an accurate emulator over three parameters can be built with as few as 39 data points.

5.1.9. The model of oceanic pelagic stoichiometry (MOPS)
The MOPS is a global ocean biogeochemical model [48] that simulates the cycling of nutrients (i.e. phosphorus, nitrogen), phytoplankton, zooplankton, dissolved oxygen and dissolved inorganic carbon. MOPS is coupled to the transport matrix method (TMM), a computational framework for efficient advective-diffusive transport of ocean biogeochemical tracers [22, 49]. In this study we use monthly mean transport matrices derived from a configuration of MITgcm [50] with a horizontal resolution of 2.8° and 15 vertical levels. There are 6 MOPS input parameters considered in this case, whose definitions and ranges are described in an optimization study by [51]. Each simulation involves integrating the model for 3000 years to a steady state starting from a uniform spatial distribution of tracers. Annual mean 3D fields of oxygen, phosphorus, and nitrate at the end of the simulation are used for training. All code and relevant data used for the simulations are freely available [49].

5.2. Super-architecture
The super-architecture employed for most cases is shown in figure 1. It consists of two fully connected layers at the beginning, followed by combinations of different types of convolutional layers. Rectified Linear Units are used as the nonlinearity. Most of the nodes contain 64 channels with a growing size of the signal from 4 to 250 at the end. For ICF JAG that has output signal of size 64×64, the size written in figure 1 is capped at 64.

After the first two fully connected layers, each pair of adjacent nodes are connected by an identity layer and a selection of multiple convolutional layers. The identity layers serve as the skip connection for each layer which is always present in every sampled architecture. The member j in group i of layers is assigned a

Table 2. List of hyperparameters used in training the emulators.

Hyperparameters	For DENSE	For manual design DNN	Notes
n_{epochs}	3000	3000	How many epochs
α_1	3.06×10^{-4}	4.34×10^{-3}	Learning rate for weight update
m_1	35	72	Size of minibatch in weight update
γ_1	0.757	0.9913	Decaying multiplier for α_1
s_1	513	7	Apply the decay multiplier for α_1 after this many update steps
α_2	4.88×10^{-3}	—	Learning rate for architectural update
m_2	142	—	Size of minibatch in architectural update
γ_2	0.701	—	Decaying multiplier for α_2
s_2	918	—	Apply the decay multiplier for α_2 after this many epochs
p_{val}	2	1	Going through the validation dataset this many times in one epoch

network parameters, b_{ij}, and the probability of member j being selected among the group is determined by the softmax function,

$$p_{ij} = \frac{\exp b_{ij}}{\sum_k \exp b_{ik}}. \tag{4}$$

Each group of layers consist of one zero layer which multiplies all the inputs to zero. This provides an option for DENSE to remove the layer and make the neural network shallower.

For two nodes with different sizes, we added modified transposed convolutional layers in the group. In the modified transposed convolutional layers, the signal is expanded just as in the normal transposed convolutional layer, but the 'holes' are filled with a trainable constant instead of zeros. The convolutional layers and identity layers between two nodes of different sizes operate by upsampling the previous node to match the size of the target node using the nearest neighbor.

5.3. Hyperparameters

The list of hyperparameters used in the algorithm are given in table 2. The hyperparameters were chosen to minimize the validation loss function for OTS using CMA-ES [14, 24]. The same hyperparameters were used for all cases.

5.4. Other emulator builder methods

In training the emulators using non-deep learning methods, we employed the scikit-learn library [25] in Python. We use the default parameters suggested in the library to build the emulators for all cases.

For models that can only predict a single output, an ensemble of models are trained to predict different outputs in one simulation. For CycleGAN in the ICF JAG and ICF JAG Scalars cases, the model was trained and obtained according to [23, 26].

5.5. Solving inverse problems with emulators

The parameter retrieval processes for XRTS and Halo to produce figure 4 were done using the CMA-ES [14] algorithm with population size 32 and 1200 maximum function evaluations. Default parameters suggested in [14]. were used. To give a fair results comparison, we used the same algorithm parameters and conditions in parameter retrievals via simulations and emulators.

For the Bayesian posterior sampling process, we employed the affine-invariant ensemble MCMC algorithm [29] with 256 walkers to collect 200 000 samples for XRTS and XES cases. The likelihood is uniform when the generated spectrum lies in a given band and it is zero when it lies outside the band. The band is 0.035 J keV^{-1} ster^{-1} in XES and 3.5% in XRTS as used in [27]. The justification of this form of likelihood is also provided in the supplementary materials of [27].

5.6. Derivation of prediction uncertainty

The randomization of the network architecture can be seen as a special case of dropout. Therefore, the derivation of the prediction uncertainty follows the derivation in Monte Carlo (MC) dropout very closely [31].

Denote the input and output from the training dataset as $\mathbf{X}$ and $\mathbf{Y}$ respectively and write $\boldsymbol{\omega}$ as the active weights in the neural network. Given the training data, $\mathbf{X}$ and $\mathbf{Y}$, the posterior distribution of the weights in the neural network can be written as

$$\mathbb{P}(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y}) = \frac{\mathbb{P}(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega})\mathbb{P}(\boldsymbol{\omega})}{Z} \tag{5}$$

where Z is the normalization constant, $\mathbb{P}(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega})$ is the likelihood of observing $\mathbf{Y}$ with weights $\boldsymbol{\omega}$, and $\mathbb{P}(\boldsymbol{\omega})$ is the prior distribution of the weights.

The posterior distribution of the weights are intractable, so we need to use variational inference in making the approximation to the posterior distribution. Let the variational distribution takes the form of $\mathbb{Q}(\boldsymbol{\omega})$ where

$$\boldsymbol{\omega_{ij}} = \mathbf{w_{ij}} z_{ij}, \text{ and} \tag{6}$$

$$z_{ij} \sim \text{Bernoulli}[p_{ij}(b_{ij})] \tag{7}$$

where $\mathbf{w_{ij}}$ is the weights of layers j in group i, p_{ij} is the probability of being selected as a function of the network variable, b_{ij}, as written in equation (4).

In order to get the best approximation of the posterior distribution $\mathbb{P}(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})$ with $\mathbb{Q}(\boldsymbol{\omega})$, the Kullback–Leibler (KL) divergence should be minimized. The KL divergence to be minimized can be expressed as

$$\text{KL} = -\int \mathbb{Q}(\boldsymbol{\omega}) \log\left[\mathbb{P}(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega})\right] \mathrm{d}\boldsymbol{\omega} + \text{KL}\left[\mathbb{Q}(\boldsymbol{\omega})||\mathbb{P}(\boldsymbol{\omega})\right]. \tag{8}$$

The integral on the first term on the right hand side can be approximated by drawing samples from $\boldsymbol{\omega_n} \sim \mathbb{Q}(\boldsymbol{\omega})$ and performing the Monte Carlo integration on the negative log-likelihood, $-\log\left[\mathbb{P}(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega_n})\right]$. The second term on the right hand side is approximated to be $\sum_{ij} \frac{p_{ij} l}{2} ||\mathbf{w_{ij}}||^2$ where l is the prior assumption of the length scale of the distribution. We can take the prior assumption of small length scale to be able to capture high variability region better and therefore making the second term small.

With various approximation above, the KL divergence in equation (8) to be minimized can be expressed as

$$\text{KL} \approx -\frac{1}{N}\sum_n \log\left[\mathbb{P}(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega_n})\right]. \tag{9}$$

By defining the negative log likelihood as the Huber loss function, we obtain that minimizing the KL divergence in the equation (9) is equivalent to minimizing the loss function in equations (1) and (2). Therefore, the optimized parameters after the training can be used to approximate the posterior distribution of the weights in the form of $\mathbb{Q}(\boldsymbol{\omega})$.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: http://doi.org/10.5281/zenodo.3782843.

Acknowledgments

M F K and S M V acknowledge support from the UK EPSRC Grant EP/P015794/1 and the Royal Society. S M V is a Royal Society University Research Fellow. G G acknowledges support from AWE plc., and the UK EPSRC (EP/M022331/1 and EP/N014472/1). E V is grateful for support from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant Agreement No. 805162). D W P and L D acknowledge funding from the Natural Environment Research Council (NERC) NE/P013406/1 (A-CURE).

Conflict of interest

The authors declare no conflict of interest.

Author contributions

M F K initiated the project with contributions from S M V. D W-P, L D, S O, P H, D H F, G G, M J, S K, J K, J T-M, and E V adapted and prepared the test cases based on real scientific cases in their respective fields. M F K designed and trained the deep neural networks and performed the analysis. M F K and S M V prepared the manuscript with contributions from D W-P, S O, P H, G G, J K, and E V.

ORCID iDs

M F Kasim https://orcid.org/0000-0002-8748-1737
E Viezzer https://orcid.org/0000-0001-6419-6848

References

[1] Greeley J, Jaramillo T F, Bonde J, Chorkendorff I B and Nørskov J K 2006 Computational high-throughput screening of electrocatalytic materials for hydrogen evolution *Nat. Mater.* **5** 909–13
[2] Lee H J *et al* 2009 X-ray Thomson-scattering measurements of density and temperature in shock-compressed beryllium *Phys. Rev. Lett.* **102** 115001
[3] Galdon-Quiroga J *et al* 2018 Beam-ion acceleration during edge localized modes in the ASDEX Upgrade tokamak *Phys. Rev. Lett.* **121** 025002
[4] Peterson J L, Humbird K D, Field J E, Brandon S T, Langer S H, Nora R C, Spears B K and Springer P T 2017 Zonal flow generation in inertial confinement fusion implosions *Phys. Plasmas* **24** 032702
[5] Kwan J, Heitmann K, Habib S, Padmanabhan N, Lawrence E, Finkel H, Frontiere N and Pope A 2015 Cosmic emulation: fast predictions for the galaxy power spectrum *Astrophys. J.* **810** 35
[6] Brockherde F, Vogt L, Li Li, Tuckerman M E, Burke K and Müller K-R 2017 Bypassing the Kohn-Sham equations with machine learning *Nat. Commun.* **8** 872
[7] Rupp M, Tkatchenko A, Müller K-R and Von Lilienfeld O A 2012 Fast and accurate modeling of molecular atomization energies with machine learning *Phys. Rev. Lett.* **108** 058301
[8] Ulyanov D, Vedaldi A and Lempitsky V 2018 Deep image prior *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* pp 9446–54
[9] Pham H, Guan M Y, Zoph B, Le Q V and Dean J 2018 Efficient neural architecture search via parameter sharing (arXiv:1802.03268)
[10] Cai H, Zhu L and Han S 2018 Proxylessnas: direct neural architecture search on target task and hardware (arXiv:1812.00332)
[11] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* pp 770–8
[12] Huber P J 1992 Robust estimation of a location parameter *Breakthroughs in Statistics* (Berlin: Springer) pp 492–518
[13] Williams R J 1992 Simple statistical gradient-following algorithms for connectionist reinforcement learning *Mach. Learn.* **8** 229–56
[14] Hansen N 2016 The CMA evolution strategy: a tutorial (arXiv:1604.00772)
[15] Gregori G, Glenzer S H, Rozmus W, Lee R W and Landen O L 2003 Theoretical model of x-ray scattering as a dense matter probe *Phys. Rev.* E **67** 026412
[16] Tzeferacos P *et al* 2018 Laboratory evidence of dynamo amplification of magnetic fields in a turbulent plasma *Nat. Commun.* **9** 591
[17] Regan S P *et al* 2013 Hot-spot mix in ignition-scale inertial confinement fusion targets *Phys. Rev. Lett.* **111** 045001
[18] Ciricosta O *et al* 2017 Simultaneous diagnosis of radial profiles and mix in NIF ignition-scale implosions via x-ray spectroscopy *Phys. Plasmas* **24** 112703
[19] Hatfield P W, Lindsay S N, Jarvis M J, Häußler B, Vaccari M and Verma A 2016 The galaxy–halo connection in the VIDEO survey at $0.5 < z < 1.7$ *Mon. Not. R. Astron. Soc.* **459** 2618–31
[20] Korenaga J and Sager W W 2012 Seismic tomography of Shatsky Rise by adaptive importance sampling *J. Geophys. Res. Solid Earth* **117** B08102
[21] Tegen I *et al* 2019 The global aerosol-climate model ECHAM6.3-HAM2.3—part 1: aerosol evaluation *Geosci. Model Dev.* **12** 1643–77
[22] Khatiwala S 2007 A computational framework for simulation of biogeochemical tracers in the ocean *Glob. Biogeochem. Cycles* **21** GB3001
[23] Anirudh R, Bremer P-T and Thiagrarjan J (USDOE National Nuclear Security Administration) 2019 Cycle consistent surrogate for inertial confinement fusion (available at: www.osti.gov//servlets/purl/1510714)
[24] Loshchilov I and Hutter F 2016 CMA-ES for hyperparameter optimization of deep neural networks (arXiv:1604.07269)
[25] Pedregosa F *et al* 2011 Scikit-learn: machine learning in python *J. Mach. Learn. Res.* **12** 2825–30
[26] Anirudh R, Thiagarajan J J, Bremer P-T and Spears B K 2020 Improved surrogates in inertial confinement fusion with manifold and cycle consistencies *Proc. Natl Acad. Sci.* **117** 9741–6
[27] Kasim M F, Galligan T P, Topp-Mugglestone J, Gregori G and Vinko S M 2019 Inverse problem instabilities in large-scale modeling of matter in extreme conditions *Phys. Plasmas* **26** 112706
[28] Wierstra D, Schaul T, Peters J and Schmidhuber J 2008 Natural evolution strategies *2008 IEEE Congress on Evolutionary Computation* (*IEEE World Congress on Computational Intelligence*) (IEEE) pp 3381–7
[29] Goodman J and Weare J 2010 Ensemble samplers with affine invariance *Commun. Appl. Math. Comput. Sci.* **5** 65–80
[30] Srivastava N, Hinton G, Krizhevsky A, Sutskever I and Salakhutdinov R 2014 Dropout: a simple way to prevent neural networks from overfitting *J. Mach. Learn. Res.* **15** 1929–58
[31] Gal Y and Ghahramani Z 2016 Dropout as a Bayesian approximation: representing model uncertainty in deep learning *Int. Conf. on Machine Learning* pp 1050–9
[32] Gal Y, Hron J and Kendall A 2017 Concrete dropout *Advances in Neural Information Processing Systems* pp 3581–90
[33] McInnes L, Healy J and Melville J 2018 UMAP: uniform manifold approximation and projection for dimension reduction (arXiv:1802.03426)
[34] Ronen B, Jacobs D, Kasten Y and Kritchman S 2019 The convergence rate of neural networks for learned functions of different frequencies *Advances in Neural Information Processing Systems* pp 4763–72
[35] Chowdhury D R and Kasim M F 2019 Efficient parameter sampling for neural network construction (arXiv:1912.10559)
[36] Kates-Harbeck J, Svyatkovskiy A and Tang W 2019 Predicting disruptive instabilities in controlled fusion plasmas through deep learning *Nature* **568** 526–31
[37] Emma C, Edelen A, Hogan M J, O'Shea B, White G and Yakimenko V 2018 Machine learning-based longitudinal phase space prediction of particle accelerators *Phys. Rev. Accel. Beams* **21** 112802
[38] Liu B, Zhang Q and Gielen G G E 2013 A Gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems *IEEE Trans. Evol. Comput.* **18** 180–92

[39] Shahriari B, Swersky K, Wang Z, Adams R P and De Freitas N 2015 Taking the human out of the loop: a review of bayesian optimization *Proc. IEEE* **104** 148–75
[40] Kritcher A L *et al* 2008 Ultrafast x-ray Thomson scattering of shock-compressed matter *Science* **322** 69–71
[41] Zohm H 1996 Edge localized modes (ELMs) *Plasma Phys. Control. Fusion* **38** 105
[42] Cavedon M *et al* 2017 Pedestal and E_r profile evolution during an edge localized mode cycle at ASDEX Upgrade *Plasma Phys. Control. Fusion* **59** 105007
[43] Viezzer E *et al* 2018 Ion heat transport dynamics during edge localized mode cycles at ASDEX Upgrade *Nucl. Fusion* **58** 026031
[44] Fable E *et al* 2013 Novel free-boundary equilibrium and transport solver with theory-based models and its validation against ASDEX Upgrade current ramp scenarios *Plasma Phys. Control. Fusion* **55** 124028
[45] Willensdorfer M *et al* 2013 Particle transport analysis of the density build-up after the L–H transition in ASDEX Upgrade *Nucl. Fusion* **53** 093020
[46] Murray S 2017 Halomod: python package for dealing with the halo model *Github* (available at: https://github.com/halomod/halomod)
[47] Wake D A *et al* 2011 Galaxy clustering in the newfirm medium band survey: the relationship between stellar mass and dark matter halo mass at $1 < z < 2$ *Astrophys. J.* **728** 46
[48] Kriest I and Oschlies A 2015 MOPS-1.0: modelling the regulation of the global oceanic nitrogen budget by marine biogeochemical processes *Geosci. Model Dev.* **8** 2929–57
[49] Khatiwala S 2018 samarkhatiwala/tmm: version 2.0 of the transport matrix method software *Zenodo* (https://doi.org/10.5281/zenodo.1246300)
[50] Marshall J, Adcroft A, Hill C, Perelman L and Heisey C 1997 A finite-volume, incompressible Navier Stokes model for studies of the ocean on parallel computers *J. Geophys. Res. Oceans* **102** 5753–66
[51] Kriest I, Sauerland V, Khatiwala S, Srivastav A and Oschlies A 2017 Calibrating a global three-dimensional biogeochemical ocean model (MOPS-1.0) *Geosci. Model Dev.* **10** 127–54