High Power Laser Science and Engineering, (2023), Vol. 11, e55, 41 pages.
doi:10.1017/hpl.2023.47

HIGH POWER LASER SCIENCE AND ENGINEERING

REVIEW

Data-driven science and machine learning methods in laser–plasma physics

Andreas Döpp[1,2], Christoph Eberle[1], Sunny Howard[1,2], Faran Irshad[1], Jinpu Lin[1], and Matthew Streeter[3]

[1] *Ludwig-Maximilians-Universität München, Garching, Germany*

[2] *Department of Physics, Clarendon Laboratory, University of Oxford, Oxford, UK*

[3] *School for Mathematics and Physics, Queen's University Belfast, Belfast, UK*

(Received 30 November 2022; revised 30 March 2023; accepted 24 May 2023)

Abstract

Laser-plasma physics has developed rapidly over the past few decades as lasers have become both more powerful and more widely available. Early experimental and numerical research in this field was dominated by single-shot experiments with limited parameter exploration. However, recent technological improvements make it possible to gather data for hundreds or thousands of different settings in both experiments and simulations. This has sparked interest in using advanced techniques from mathematics, statistics and computer science to deal with, and benefit from, big data. At the same time, sophisticated modeling techniques also provide new ways for researchers to deal effectively with situation where still only sparse data are available. This paper aims to present an overview of relevant machine learning methods with focus on applicability to laser-plasma physics and its important sub-fields of laser-plasma acceleration and inertial confinement fusion.

Keywords: deep learning; laser–plasma interaction; machine learning

1. Introduction

1.1. Laser–plasma physics

Over the past decades, the development of increasingly powerful laser systems[1,2] has enabled the study of light–matter interaction across many regimes. Of particular interest is the interaction of intense laser pulses with plasma, which is characterized by strong nonlinearities that occur across many scales in space and time[3,4]. These laser–plasma interactions are of interest both for fundamental physics research and as emerging technologies for potentially disruptive applications.

Regarding fundamental research, high-power lasers have, for instance, been used to study transitions from classical electrodynamics to quantum electrodynamics (QED) via the radiation reaction, where a particle's backreaction to its radiation field manifests itself in an additional force[5–7]. Recent proposals to extend intensities to the Schwinger limit[8], where the electric field strength of the light is comparable to the Coulomb field, could allow the study of novel phenomena expected to occur due to a breakdown of perturbation theory. In an only slightly less extreme case, high-energy density physics (HEDP)[9] research uses lasers for the production and study of states of matter that cannot be reached otherwise in terrestrial laboratories. This includes creating and investigating material under extreme pressures and temperatures, leading to exotic states such as warm–dense matter[10–12].

Apart from the fundamental interest, there is also considerable interest in developing novel applications that are enabled by these laser–plasma interactions. Two particularly promising application areas have emerged over the past decades, namely the production of high-energy radiation beams (electrons, positrons, ions, X-rays, gamma-rays) and laser-driven fusion.

Laser–plasma acceleration (LPA) aims to accelerate charged particles to high energies over short distances by inducing charge separation in the plasma, for example, in the form of plasma waves to accelerate electrons or by stripping electrons from thin-foil targets to accelerate ions.

Correspondence to: Andreas Döpp, Ludwig-Maximilians-Universität München, Am Coulombwall 1, 85748 Garching, Germany. Email: a.doepp@lmu.de

The former scenario is called laser wakefield acceleration (LWFA)[13–15]. Here a high-power laser propagates through a tenuous plasma and drives a plasma wave, which can take the shape of a spherical ion cavity directly behind the laser. The fields within this so-called bubble typically reach around 100 GV/m, allowing LWFA to accelerate electrons from rest to GeV energies within centimeters[16–21]. While initial experiments were single-shot in nature and with significant shot-to-shot variations, the performance of LWFA has drastically increased in recent years. Particularly worth mentioning in this regard are the pioneering works on LWFA at the kHz-level repetition rate, starting with an early demonstration in 2013[22] and in the following years mostly developed by Faure *et al.*[23], Rovige *et al.*[24] and Salehi *et al.*[25], as well as the day-long stable operation achieved by Maier *et al.*[26]. As a result of these efforts, typical experimental datasets in publications have significantly increased in size. To give an example, first studies on the so-called beamloading effect in LWFA were done on sets of tens of shots[27], whereas newer studies include hundreds of shots[28] or, most recently, thousands of shots[29].

Laser-driven accelerators can also function as bright radiation sources via the processes of bremsstrahlung emission[30,31], betatron radiation[32,33] and Compton scattering[34–36]. These sources have been used for a variety of proof-of-concept applications[37], ranging from spectroscopy studies of warm–dense matter[10] and over imaging[38,39] to X-ray computed tomography (CT)[40–43]. It has also recently been demonstrated that LWFA can produce electron beams with sufficiently high beam quality to drive free-electron lasers (FELs)[44,45], offering a potential alternative driver for next generation light sources[46].

Laser-ion acceleration[47,48] uses similar laser systems as LWFA, but typically operates with more tightly focused beams to reach even higher intensities. Here the goal is to separate a population of electrons from the ions and then use this electron cloud to strip ions from the target. The ions are accelerated to high energies by the fields that are generated by the charge separation process. This method has been used to accelerate ions to energies of a few tens of MeV/u in recent years. In an alternative scheme, radiation pressure acceleration[49,50], the laser field is used to directly accelerate a target. Even though it uses the same or similar lasers, ion acceleration typically operates at much lower repetition rate because of its thin targets, which are not as easily replenished as gas-based plasma sources used in LWFA. Recent target design focuses on the mitigation of this issue, for instance using cryogenic jets[51–54] or liquid crystals[55].

Another application of potentially high societal relevance is laser-driven inertial confinement fusion (ICF)[56,57], where the aim is to induce fusion by heating matter to extremely high temperatures through laser–plasma interaction. As the name suggests, confinement is reached via the inertia of the plasma, which is orders of magnitude larger than the thermal energy. To achieve spatially homogenous heating that can penetrate deep into the fusion target, researchers commonly resort to driving the ignition process indirectly. In this method, light is focused into an empty cylindrical cavity, called a hohlraum, which is used to radiate a nearly isotropic blackbody spectrum that extends into the X-ray regime and is subsequently absorbed by the imploding capsule[58]. In contrast to this, direct-drive methods aim to directly drive the thermonuclear fusion process[59]. In this case, the laser is focused directly onto the fuel capsule. Direct-drive poses some challenges that are not present in indirect-drive schemes. For instance, the light has to penetrate through the high-density plasma shell surrounding the capsule and the illumination is less homogeneous, because of which the compressed target can be subject to large hydrodynamic instabilities. Advanced ignition schemes aim to separate compression of the thermonuclear fuel from triggering the ignition process. Examples are fast ignition[60], which uses the high-intensity laser pulse to directly heat the compressed and dense fusion target, and shock ignition[61], which uses a shock wave to compress the target. Recently, the first ICF experiment at the National Ignition Facility (NIF) has reported reaching the burning-plasma state via the combination of indirect-drive with advanced target design[62]. This breakthrough has re-enforced scientific and commercial interest in ICF, which is now also being pursued by a number of start-up companies.

1.2. Why data-driven techniques?

In recent years, data-driven methods and machine learning have been applied to a wide range of physics problems[63], including for instance quantum physics[64,65], particle physics[66], condensed matter physics[67], electron microscopy[68] and fluid dynamics[69]. In comparison, its use in laser–plasma physics is still in its infancy and is curiously driven by both data abundance and data scarcity. Regarding the former, fast developments in both laser technology[70] and plasma targetry[71–74] nowadays permit the operation of laser–plasma experiments – in particular laser–plasma accelerators – at joule-level energies and multi-Hz to kHz repetition rates[75–79]. The vast amounts of data generated by these experiments can be used to develop data-driven models, which are then employed in lieu of conventional theoretical or empirical approaches, or to augment them. In contrast, laser systems used for inertial fusion research produce MJ-level laser pulses and operate at repetition rates as low as one shot per day. With such a sparse number of independent experimental runs, data-driven models are used to extract as much information as possible from the existing data or combine them with other information sources, such as simulations.

The success of all of the above applications depends on the precise control of a complex nonlinear system. In order to

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

optimize and control the process of laser–plasma interaction, it is essential to understand the underlying physics and to be able to model the complex plasma response to an applied laser pulse. However, this is complicated by the fact that it is a strongly nonlinear, multi-scale, multi-physics phenomenon. While analytical and numerical models have been essential tools for understanding laser–plasma interactions, they have several limitations. Firstly, analytical models are often limited to low-order approximations and therefore cannot accurately predict the behavior of complex laser–plasma systems. Secondly, accurate numerical simulations require immense computational resources, often millions of core hours, which limits their use for optimizing and controlling real-world laser–plasma experiments. In addition, the huge range of temporal and spatial scales means that in practice many physical processes (ionization, particle collisions, etc.) can only be treated approximately in large-scale numerical simulations. Because of this, one active area of research is to automatically extract knowledge from data in order to build faster computational models that can be used for prediction, optimization, classification and other tasks.

Another important problem is that the diagnostics employed in laser–plasma physics experiments typically only provide incomplete and insufficient information about the interaction, and key properties must be inferred from the limited set of available observables. Such inverse problems can be hard to solve, especially when some information is lost in the measurement process and the problem becomes ill-posed. Modern methods, such as compressed sensing (CS) and deep learning, are strong candidates to facilitate the solution of such problems and thus retrieve so-far elusive information from experiments.

The goal of this review is to summarize the rapid, recent developments of data-driven science and machine learning in laser–plasma physics, with particular emphasis on LPA, and to provide guidance for novices regarding the tools available for specific applications. We would like to start with a disclaimer that the lines between machine learning and other methods are often blurred.[1] Given the multitude of often competing subdivisions, we have chosen to organize this review based on a few broad classes of problems that we believe to have highest relevance for laser–plasma physics and its applications. These problems are modeling and prediction (Section 2), inverse problems (Section 3), optimization (Section 4), unsupervised learning for data analysis (Section 5) and, lastly, image analysis with supervised learning techniques (Section 6). Partial overlap between these applications and the tools used is unavoidable and is where possible indicated via cross-references. This is particularly true in the case of neural networks, which have found a broad usage across applications. Each section includes introductions to the most common techniques that address the problems outlined above. We explicitly include what can be considered as 'classical' data-driven techniques in order to provide a better context for more recent methods. Furthermore, examples for implementations of specific techniques in laser–plasma physics and related fields are highlighted in separate text boxes. We hope that these will help the reader to get a better idea of which methods might be most adequate for their own research. An overview of the basic application areas is shown in Figure 1.

To maintain brevity and readability, some generalizations and simplifications are made. For detailed descriptions and strict definitions of methods, the reader may kindly refer to the references given throughout the text. Furthermore, we would like to draw the reader's attention to some recent reviews on the application of machine learning techniques in the related fields of plasma physics[80–82], ultra-fast optics[83] and HEDP[84].

2. Modeling and prediction

Many real-life and simulated systems are expensive to evaluate in terms of time, money or other limited resources. This is particularly true for laser–plasma physics, which either hinges on the limited access to high-power laser facilities or requires high-performance computing to accurately model the ultra-fast laser–plasma interaction. It is therefore desirable to find models of the system, sometimes called digital twins[85], which are comparatively cheap to evaluate and whose predictions can be used for extensive analyses. In engineering and especially in the context of model-based optimization (see Section 4), such lightweight models are often referred to as surrogate models. Reduced-order models feature fewer degrees of freedom than the original system, which is often achieved using methods of dimensionality reduction (see Section 5.3).

[1] For instance, deep learning is a sub-field of machine learning where deep neural networks are used, and this term is nowadays often used interchangeably with neural networks. Similarly, data-driven science is sometimes used instead of machine learning, but also includes a variety of methods from computer science, applied mathematics and statistics, as well as data science, which is a field in itself. Even within itself, machine learning is a heavily segmented research field, whose community can famously be divided into five 'tribes', namely symbolists, connectionists, evolutionists, Bayesians and analogizers. Each of these groups has pioneered different tools, all of which have in recent years experienced a resurgence in popularity. The arguably most popular branch is connectionism, which focuses on the use of artificial neural networks. However, some challenges seen in laser–plasma physics require different approaches and, for instance, Bayesian optimization has recently drawn considerable research interest. Another line of division is often drawn between supervised and unsupervised methods. While supervised methods are usually trained from known datasets to build a model that can classify new data, unsupervised methods attempt to find some structure in the data without such pre-existing knowledge. Alternatively, one can distinguish between online and batch methods, where the former learn from data as it becomes available, and can therefore be used in an experimental setting, while the latter require access to the full dataset before learning can begin. Yet another important distinction can be made between parametric and non-parametric methods, where the former rely on a set of parameters that is fixed and known in advance, while the latter do not make this assumption, but rather learn the model parameters from the data.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

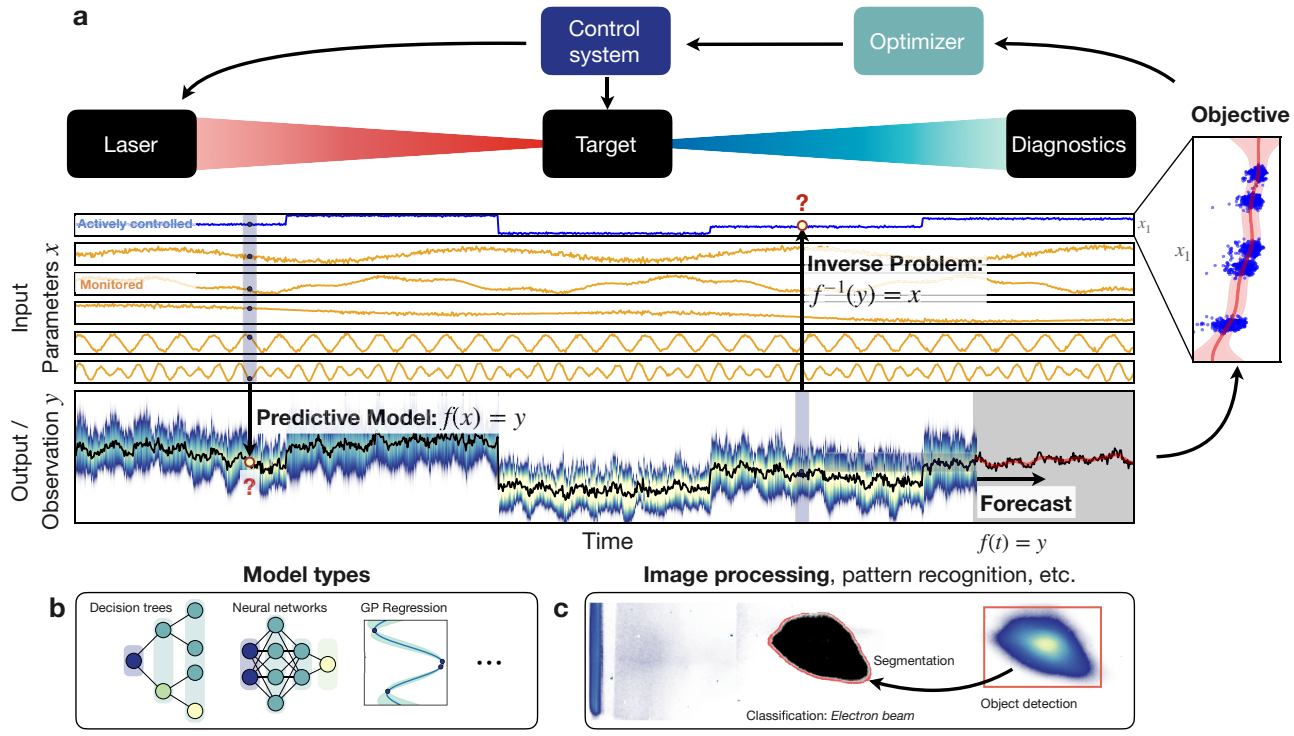

Figure 1. Overview of some of the machine learning applications discussed in this manuscript. (a) General configuration of laser–plasma interaction setups, applicable to both experiments and simulations. The system will have a number of input parameters of the laser and target. Some of these are known and actively controlled (e.g., laser energy, plasma density), some are monitored and others are unknown and essentially contribute as noise to the observations. Predictive models take the known input parameters x and use some models to predict the output y. These models are discussed in Section 2.1 and some of them are sketched in (b). Inversely, in some cases one will want to derive the initial conditions from the output. These inverse problems are discussed in Section 3. In other cases one might be interested in a temporal evolution, discussed in Section 2.2. The output from observations or models can be used to optimize certain objectives, which can then be fed back to the control system to adjust the input parameters (see Section 4). Observations may also require further processing, for example, the image processing in (c) to detect patterns or objects. Note that sub-figure (a) is for illustrative purposes only and based on synthetic data.

The general challenge in modeling is to find a good approximation of $f^*(x)$ for the real system $f(x)$ based on only a limited number of examples $f(x_n) = y_n$, the training data. Here x_n is an n-sized set of vector-valued input parameters and y_n is the corresponding output. To complicate things further, any real measurement will be subject to noise, so y_n has to be interpreted as a combination of the true value and some random noise contribution. Another complication arises from having imperfect or unrepeatable controllers for the input parameters x_n. This can result in having different output values for the same set of input parameters.

The predictive models discussed in Section 2.1 below are mostly used to interpolate between training data, whereas the related problem of forecasting (Section 2.2) explicitly deals with the issue of extrapolating from existing data to new, so-far unseen data. In Section 2.3 we briefly discuss how models can be used to provide direct feedback for laser–plasma experiments.

2.1. Predictive models

In this section, we describe some of the most common ways to create predictive models, starting with the 'classic' approaches of spline interpolation and polynomial regression, before discussing some modern machine learning techniques.

2.1.1. Spline interpolation

The simplest way of constructing a model for a system, be it a real-world system or some complex simulation, is to use a set of n training points and to predict that every unknown position will have the same value as the nearest neighboring training point (see Figure 2(a)). A straightforward extension with slightly higher computational requirements is to connect the training points with straight lines, resulting in a piecewise linear function. Both methods, however, are not continuously differentiable, and for instance less suited for the integration of the model into an optimization process (see Section 4). A more advanced approach to interpolating the training points is to use splines (see Figure 2(b)), which require the piecewise-interpolated functions to agree up to a certain derivative order. For instance, cubic splines are continuously differentiable up to the second derivative. While higher-order spline interpolation works well in 1D cases, it becomes increasingly difficult in multi-dimensional settings. Furthermore, the interpolation approach does not

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

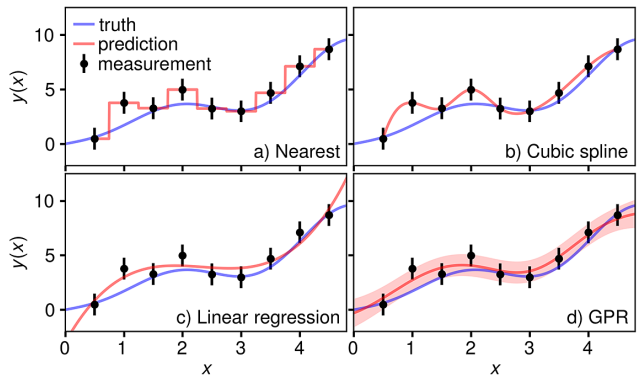

Figure 2. Illustration of standard approaches to making predictive models in machine learning. The data were sampled from the function $y = x\left(1+\sin x^2\right)+\epsilon$ with random Gaussian noise, ϵ, for which $\left\langle \epsilon^2 \right\rangle = 1$. The data have been fitted by (a) nearest neighbor interpolation, (b) cubic spline interpolation, (c) linear regression of a third-order polynomial and (d) Gaussian process regression.

allow for incorporating uncertainty or stochasticity, which is present in real-world measurements. Therefore, it will treat noise components as a part of the system, including, for instance, outliers.

2.1.2. *Regression*

In some specific cases the shortcomings of interpolation approaches can be addressed by using regression models. For instance, simple systems can often be described using a polynomial model, where the coefficients of the polynomial are determined via a least-squares fit (see Figure 2(c)), that is, minimizing the sum of the squares of the difference between the predicted and observed data. The results are generally improved by including more terms in the polynomial but this can lead to overfitting – a situation where the model describes the noise in the data better than the actual trends, and consequently will not generalize well to unseen data. Regression is not restricted to polynomials, but may use all kinds of mathematical models, often motivated by a known underlying physics model. Crucially, any regression model requires the prior definition of a function to be fitted, thus posing constraints on the relationships that can be modeled. In practice this is one of the main problems with this approach, as complex systems can scarcely be described using simple analytical models. Before using these models, it is thus important to verify their validity, for example, using a measure for the goodness of fit such as the correlation coefficient R^2, the χ^2 test or the mean-squared error (MSE).

2.1.3. *Probabilistic models*

The field of probabilistic modeling relies on the assumption that the relation between the observed data and the underlying system is stochastic in nature. This means that the observed data are assumed to be drawn from a probability distribution for each set of input parameters to a generative model. Inversely, one can use statistical methods to infer the parameters that best explain the observed data. We will discuss such *inference* problems in more detail in the context of solving (ill-posed) inverse problems in Section 3.2.

Probabilistic models can generally be divided into two methodologies, frequentist and Bayesian. At the heart of the frequentist approach lies the concept of likelihood, $p\left(y|\theta\right)$, which is the probability of some outcomes or observations y given the model parameters θ (see, e.g., Ref. [86] for an introduction). A model is fitted by maximum likelihood estimation (MLE), which means finding the model parameters $\widehat{\theta}$ that maximize the likelihood, $p\left(y|\widehat{\theta}\right)$. When observations $y = (y_1, y_2, \dots)$ are independent, the probability of observing y is the product of the probabilities of each observation, $p(y) = \prod_{i=1}^{n} p\left(y_i\right)$. As sums are generally easier to handle than products, this is often expressed in terms of the log-likelihood function that is given by the sum of the logarithms of each observation's probability, that is

$$\log p\left(y|\theta\right) = \sum_{i=1}^{n} \log p\left(y_i|\theta\right). \tag{1}$$

Probabilities have values between 0 and 1, so the log-likelihood is always negative and, the logarithm being a strictly monotonic function, minimizing the log-likelihood maximizes the likelihood:

$$\widehat{\theta}_{\mathrm{MLE}} = \arg\max_{\theta} \left\{p\left(y|\theta\right)\right\} = \arg\min_{\theta} \left\{\log p\left(y|\theta\right)\right\}. \tag{2}$$

The optimum can be found using a variety of optimization methods, for example, gradient descent, which are described in more detail in Section 4. A simple example is the use of MLE for parameter estimation in regression problems. In the case in which the error $(Ax - y)$ is normally distributed, this turns out to be equivalent to the least-squares method we discussed in the previous section.

The MLE is often seen as the simplest and most practical approach to probabilistic modeling. One advantage is that it does not require any *a priori* assumptions about the model parameters, rather only about the probability distributions of the data. However, this can also be a drawback if useful prior knowledge of the model parameters is available. In this case one would turn to Bayesian statistics. Here one assesses information about the probability that a hypothesis about the parameter values θ is correct given a set of data x. In this context the probabilistic model is viewed as a collection of conditional probability densities $p\left(\theta|y\right)$ for each set of observed data y, with the aim of finding the posterior distribution $p\left(\theta|y\right)$, that is, the probability of some parameters given the data observations. This can be done by applying Bayes' rule, as follows:

$$p\left(\theta|y\right) = \frac{p\left(y|\theta\right)p\left(\theta\right)}{p(y)}, \tag{3}$$

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

where $p(y|\theta)$ is known from above as the likelihood function and $p(\theta)$ denotes the prior distribution, that is, our *a priori* knowledge about the parameters before we observe any data y. The denominator, $p(y) = \int p\left(y|\theta'\right) \cdot p\left(\theta'\right) \mathrm{d}\theta'$, is called the evidence and ensures that both sides of Bayes' rule are properly normalized by integrating over all possible input parameters θ. Once the posterior distribution is known, we can maximize it and get the maximum a posteriori (MAP) estimate:

$$\widehat{\theta}_{\mathrm{MAP}} = \arg\max_{\theta} \{p(\theta|y)\}. \tag{4}$$

As mentioned above, a particular strength of the Bayesian approach is that we can encode *a priori* information in the prior distribution. Taking the example of polynomial regression, we could, for instance, set *a priori* distribution $p(\theta)$ for the regression coefficients θ that favors small coefficients, thus penalizing high-order polynomials. Another advantage of using the Bayesian framework is that one can quantify the uncertainty in the model result. This is particularly simple to compute in the special case of Gaussian distributions and their generalization, Gaussian processes (GPs), which we are going to discuss in the next section.

2.1.4. *Gaussian process regression*

A popular version of Bayesian probabilistic modeling is GP regression[87,88]. This kind of modeling was pioneered in geostatistics in the context of mining exploration and is historically also referred to as kriging, after the South African engineer Danie G. Krige, who invented the method in the 1950s. Conceptually, one can think of it as loosely related to the spline interpolation method, as it is also locally 'interpolates' the training points. Compared to splines and conventional regression methods, kriging has a number of advantages. Being a regression method, kriging can deal with noisy training points, as seen in experimental data. At the same time, the use of GPs involves minimal assumptions and can in principle model any kind of function. Lastly, the 'interpolation' is done in a probabilistic way, that is, a probability distribution is assigned to the function values at unknown positions (see Figure 2(d)). This allows for quantifying the uncertainty of the prediction. These features make kriging an attractive method for the construction of surrogate models for complex systems for which only a limited number of the function evaluations is possible, for example, due to the long runtime of the system or the high costs of the function evaluations. GP regression forms the backbone of most implementations of Bayesian optimization (BO), which we will discuss in Section 4.5, including examples for potential use cases.

Mathematically speaking, a GP is an infinite collection of normal random variables, where each finite subset is jointly normally distributed. The mean vector and covariance matrix of the multivariate normal distribution are thereby generalized to the mean function $\mu(x)$ and the covariance function $\sigma\left(x,x'\right)$, respectively, where we use the short-hand notation $x = (x_1, x_2, \dots)$ to denote the function inputs as a vector of real numbers.

A GP can be written in the following form:

$$f(x) \sim \mathcal{GP}\left(\mu(x), \sigma\left(x,x'\right)\right), \tag{5}$$

denoting that the random function $f(x)$ follows a GP with mean function $\mu(x)$ and covariance function $\sigma\left(x,x'\right)$.

The mean function $\mu(x)$ is defined as the expectation of the GP, that is, $\mu(x) = \langle f(x) \rangle$, whereas the covariance function is defined as $\sigma\left(x,x'\right) = \left\langle f(x) - \mu(x) \left(f\left(x'\right) - \mu\left(x'\right)\right)\right\rangle$. Note that in the special case of the constant mean function $\mu(x) = 0$ and a constant diagonal covariance function $\sigma\left(x,x'\right) = \sigma^2 \delta\left(x - x'\right)$, the GP simply reduces to a set of a normal random variable with zero mean and variance σ^2 commonly referred to as white noise. The covariance function is also referred to as the kernel. Its value at locations x and x' is proportional to the correlation between the function values $f(x)$ and $f\left(x'\right)$. A common choice for the covariance function is the squared exponential function, also known as the radial basis function (RBF) kernel:

$$\sigma\left(x,x'\right) = \exp\left(-\frac{\left(x - x'\right)^2}{2\ell^2}\right), \tag{6}$$

where ℓ is the length scale parameter, which controls the rate at which the correlation between $f(x)$ and $f\left(x'\right)$ decays. This kernel hyperparameter can usually be optimized by using the training data to minimize the log marginal likelihood.

It is possible to encode prior knowledge by choosing a specialized kernel that imposes certain restrictions on the model. A variety of such kernels exist. For instance, the periodic kernel (also known as exp-sine-square kernel) given by the following:

$$\sigma\left(x,x'\right) = \exp\left(-\frac{2\sin^2\left(\pi d\left(x,x'\right)/\lambda\right)}{\ell^2}\right), \tag{7}$$

with the Euclidean distance $d\left(x,x'\right)$, length scale ℓ and periodicity λ as free hyperparameters, is particularly suitable to model systems that show an oscillatory behavior.

A useful property of kernels is the generation of new kernels through the addition or multiplication of existing kernels[89]. This property provides another way to leverage prior information about the form of the function to increase the predictive accuracy of the model. For instance, measurement errors incorporated into the model by adding a Gaussian white noise kernel, as for instance done in Figure 2.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

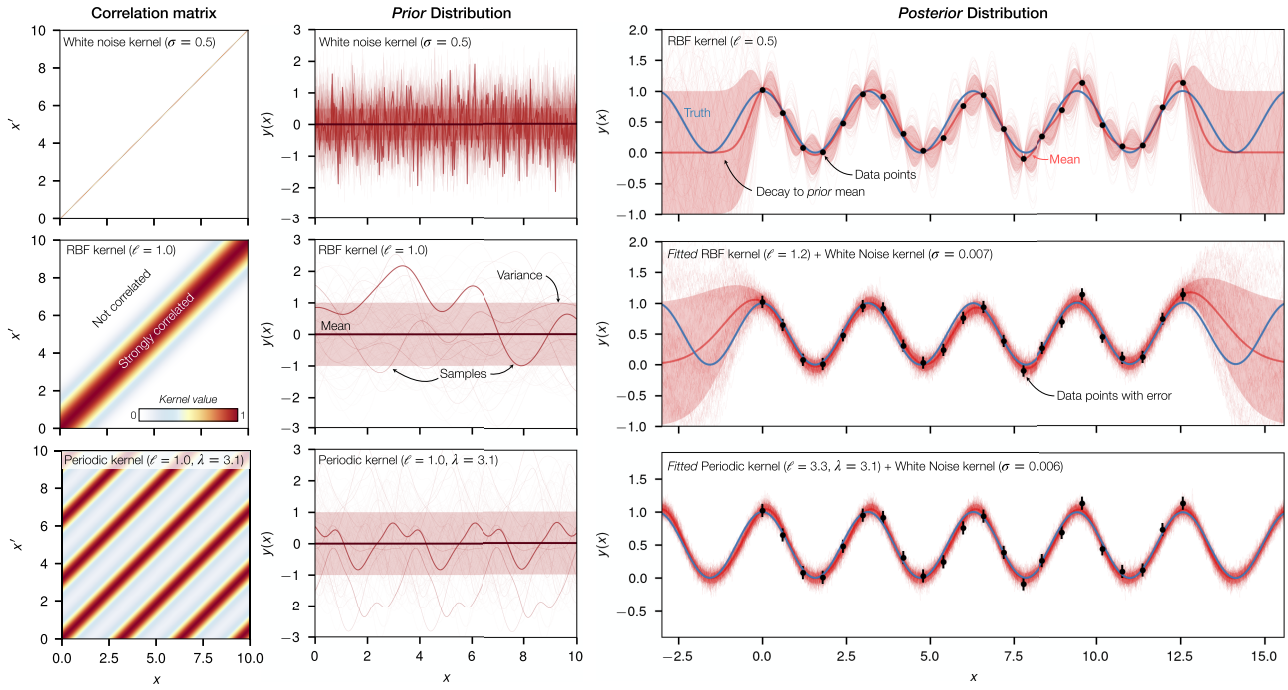

Figure 3. Gaussian process regression: illustration of different covariance functions, prior distributions and (fitted) posterior distributions. *Left:* correlation matrices between two values x and x' using different covariance functions (white noise, radial basis function and periodic). *Center:* samples of the prior distribution defined by the prior mean $\mu(x) = 0$ and the indicated covariance functions. Note that the sampled functions are depicted with increasing transparency for visual clarity. *Right:* posterior distribution given observation points sampled from $y = \cos^2 x + \epsilon$, where ϵ is random Gaussian noise with $\sigma_\epsilon = 0.1$. Note how the variance between observations increases when no noise term is included in the kernel (top row). Within the observation window the fitted kernels show little difference, but outside of it the RBF kernel decays to the mean $\mu = 0$ dependent on the length scale ℓ. This can be avoided if there exists prior knowledge about the data that can be encoded in the covariance function, in this case periodicity, as can be seen in the regression using a periodic kernel.

Figure 3 visualizes the covariance functions and their influence on the result of GP regression. This includes correlation matrices for the three covariance functions discussed above (white noise, RBF and periodic kernels). Once the mean and the covariance functions are fully defined, we can use training points for regression, that is, fit the GP and kernel parameters to the data and obtain the posterior distribution (see the right-hand panel of Figure 3).

2.1.5. *Decision trees and forests*

A decision tree is a general-purpose machine learning method that learns a tree-like model of decision rules from the data to make predictions[90]. It works by splitting the dataset recursively into smaller groups, called nodes. Each node represents a decision point, and the tree branches out from the node according to the decisions that are made. The leaves of the tree represent the final prediction; see Figure 4 for an illustration. This can be either a categorical value in classification tasks (see Section 6) or a numerical value prediction in regression tasks. An advantage of this approach is that it can learn nonlinear relationships in data without having to specify them.

To generate a decision tree one starts at the root of the tree and determines a decision node such that it optimizes an underlying decision metric, such as the MSE in regression settings or entropy and information gain in a classification setting. At each decision point the dataset is split and subsequently the metric is re-evaluated for the resulting groups, generating the next layer of decision nodes. This process is repeated until the leaves are reached. The more decision layers are used, called the depth of the tree, the more complex relationships can be modeled.

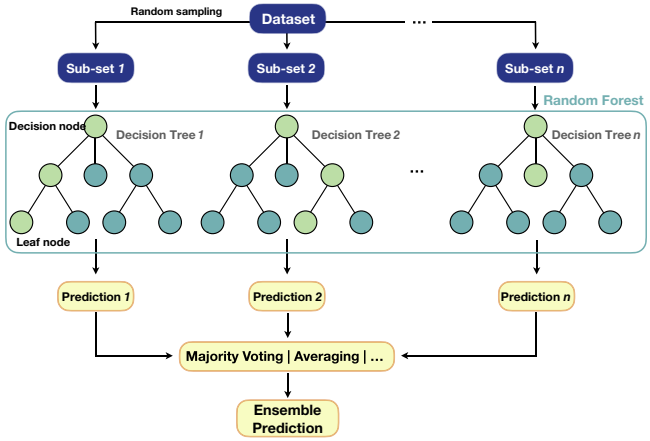

Figure 4. Sketch of a random forest, an architecture for regression or classification consisting of multiple decision trees, whose individual predictions are combined into an ensemble prediction, for example, via majority voting or averaging.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

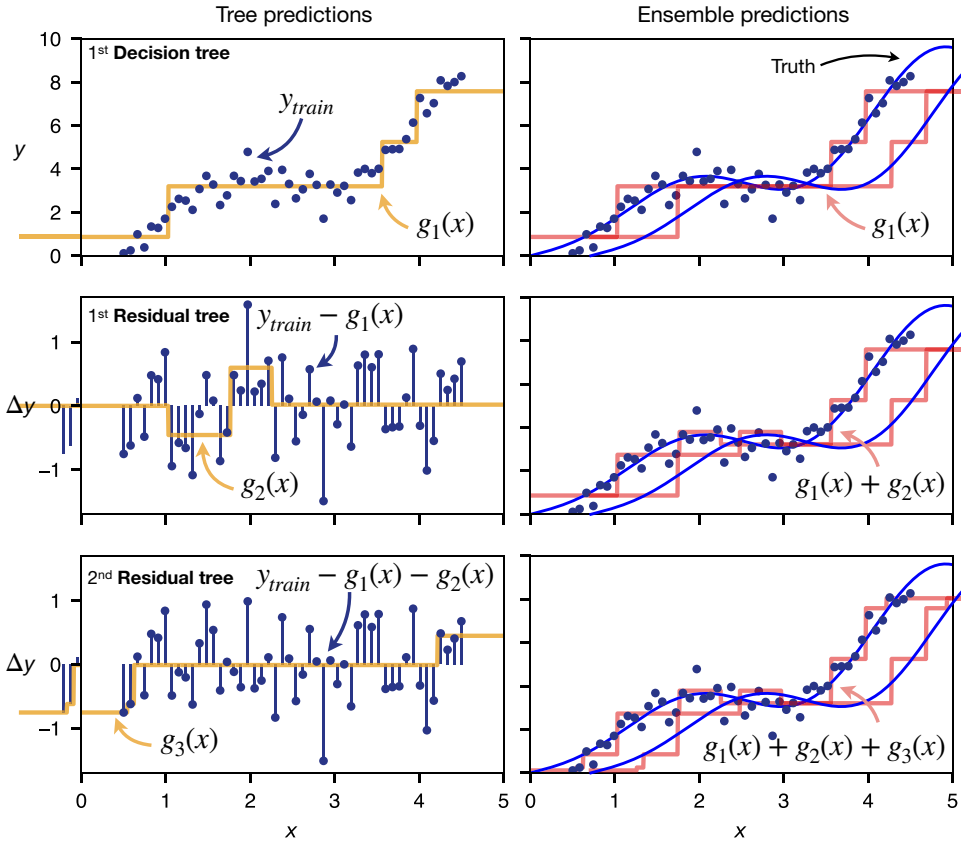

Figure 5. Example of gradient boosting with decision trees. Firstly, a decision tree g_1 is fitted to the data. In the next step, the residual difference between training data and the prediction of this tree is calculated and used to fit a second decision tree g_2. This process is repeated n times, with each new tree g_n learning to correct only the remaining difference to the training data. Data in this example are sampled from the same function as in Figure 2 and each tree has a maximum depth of two decision layers.

Decision trees are easy to implement and can provide accurate predictions even for large datasets. However, with an increasing number of decision layers they may become computationally expensive and may overfit the data, the latter being in particular a problem with noisy data. One method to address overfitting is called pruning, where branches from the tree that do not improve the performance are removed. Another effective method is to use decision-tree-based ensemble algorithms instead of a single decision tree.

One example of such an algorithm is the random forest, an ensemble algorithm that uses bootstrap aggregating or bagging to fit trees to random subsets of the data and the predictions of individually fitted decision trees are combined by majority vote or average to obtain a more accurate prediction. Another type of ensemble algorithms is boosting, where the trees are trained sequentially and each tries to correct its predecessor. A popular implementation is AdaBoost[91], where the weights of the samples are changed according to the success of the predictions of the previous trees. Gradient boosting methods[92,93] also use the concept of sequentially adding predictors, but while AdaBoost adjusts weights according to the residuals of each prediction, gradient boosting methods fit new residual trees to the remaining differences at each step (see Figure 5 for an example).

Compared to other machine learning algorithms, a great advantage of a decision tree is its explicitness in data analysis, especially in nonlinear high-dimensional problems. By splitting the dataset into branches, the decision tree naturally reveals the importance of each variable regarding the decision metric. This is in contrast to 'black-box' models, such as those created by neural networks, which must be interpreted by post hoc analysis.

Decision trees can also be used to seed neural networks by giving the initial weight parameters to train a deep neural network. Examples of using a decision tree as an initializer are the deep jointly-informed neural networks (DJINNs) developed by Humbird *et al.*[94], which have been widely applied in the high-power laser community, especially in analyzing ICF datasets. The algorithm first constructs a tree or a random forest with the tree depth set as a tunable hyperparameter. It then maps the tree to a neural network, or maps the forest to an ensemble of networks. The structure of the network (number of

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

neurons and hidden layer, initial weights, etc.) reflects the structure of the tree. The neural network is then trained using back-propagation. The use of decision trees for initialization largely reduces the computational cost while maintaining comparable performance to optimized neural network architectures. The DJINN algorithm has been applied to several classification and regression tasks in high-power laser experiments, such as ICF[95–97] and LWFA[98].

2.1.6. Neural networks

Neural networks offer a versatile framework for fitting of arbitrary functions by training of a network of connected nodes or neurons, which are loosely inspired by biological neurons. In the case of a fully connected neural network, historically called a multilayer perceptron, the inputs to one node are the outputs of all of the nodes from the preceding layer. The very first layer is simply all of the inputs for the function to be modeled, and the very last layer is the function outputs. One of the very attractive properties of neural networks is the capacity to have many outputs and inputs, each of which can be multi-dimensional. The weighting of each connection w_i and the bias for each node b are the parameters of the network, which must be trained to provide the best approximation of the function to be modeled. Each node gets activated depending on both weights and biases according to a pre-defined activation function. The nonlinear properties of activation functions are widely seen as the key properties to allow neural networks to model arbitrary functions and achieve general learning properties.

One of the simplest, but also most common, activation functions is the rectified linear unit (ReLU), which is defined as follows:

$$\mathrm{ReLU}(x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

The function argument $x = \sum x_i w_i + b$ is the sum of the weights of incoming connections, multiplied with their values x_i, and the bias of the node. The ReLU function is easy to compute, and it has the main advantage that it is linear for $x > 0$, which greatly simplifies the gradient calculation in training (see the next paragraph). Drawbacks are that it is not differentiable at $x = 0$, it is unbounded, and since it yields a constant value below 0, it has the potential to produce 'dead' neurons. The last issue is solved in the so-called leaky ReLU, which uses a reduced gradient for $x < 0$, giving an output of αx where $0 < \alpha < 1$. Other common activation functions include the logistic or sigmoid function, $\mathrm{sig}\,(x) = (1 + e^{-x})^{-1}$, and the hyperbolic tangent function, $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$.

The training is performed iteratively by passing corresponding input–output pairs to the network and comparing the true outputs to those given by the network to calculate the loss function. This is often chosen to be the MSE between the model output and the reference from the training data, but many other types of loss functions are also used (see Section 4.1.1) and the choice of loss function strongly affects the model's training. The loss is then used to modify the weights and biases via an algorithm known as 'back-propagation'[99]. Here the gradient of the loss function is calculated with respect to the weights and biases via the chain rule. One can then optimize the parameters using, for instance, stochastic gradient descent (SGD) or Adam (adaptive moment estimation)[100].

A number of hyperparameters are used to control the training process. Typical ones include the following: the number of epochs – the number of times the model sees each data point; the batch size – the number of data points the model sees before updating its gradients; and the learning rate – a factor determining the magnitude of the update to the gradient. Each factor can have a critical effect on the training of the model.

In the training process, the weights and biases are optimized in order to best approximate the function for converting the inputs into the corresponding outputs. The resulting model will yield an approximation for the unknown function $f(x)$. However, learning via this method only optimizes how well the model reproduces the training data and, in the worst case, the network essentially 'memorizes' the training data, and learns little about the desired function $f(x)$. To quantify this, a subset of the data is kept separate and used solely for validating the model. Ideally, the loss should be similar on both training and validation data, and if the model has a significantly smaller loss on the training data than on the validation set it is said to overfit, which signifies that the model has learnt the random noise of the training set. The validation loss thus quantifies how well this model generalizes and, with it, how useful it is for predictions on unseen data.

Common methods to combat overfitting include early stopping, that is, terminating the training process once the validation loss stagnates; or the use of dropout layers to randomly switch off some fraction of the network connections for each batch of the training process, thereby preventing the network from learning random noise by reducing its capacity. A further approach is to incorporate variational layers into the network. In these layers, pairs of nodes are used that represent the mean μ and standard deviation σ of a Gaussian distribution. During training, output values are sampled from this distribution and then passed on the subsequent layers, thereby requiring the network to react smoothly to these small random variations to achieve a small training loss. Both dropout layers and variational layers provide regularization that smooths the network response and prevents single nodes from dominating, resulting in improved interpolation performance and better validation

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

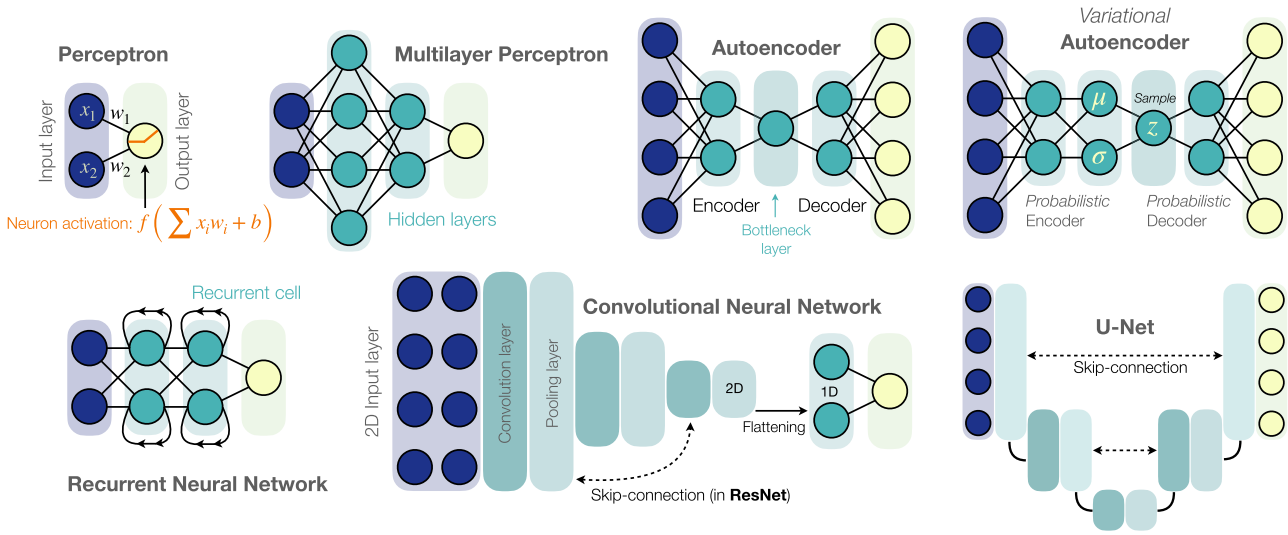

Figure 6. Simplified sketch of some popular neural network architectures. The simplest possible neural network is the perceptron, which consists of an input, which is fed into the neuron that processes the input based on the weights, an individual bias and its activation function. Multiple such layers can be stacked within so-called hidden layers, resulting in the popular multilayer perceptron (or fully connected network). Besides the direct connection between subsequent layers, there are also special connections common in many modern neural network architectures. Examples are the recurrent connection (which feeds the output of the current layer back into the input of the current layer), the convolutional connection (which replaces the direct connection between two layers by the convolutional operation) and the residual connection (which adds the input to the output of the current layer; note that the above illustration is simplified and the layers should be equal in size).

loss[101,102] (see Section 3.3 for a brief general explanation of regularization).

Beyond the classical multilayer perceptron network, there exists a plethora of different neural network architectures, as partially illustrated in Figure 6, which are suited to different tasks. In particular, convolutions layers, which extract relevant features from one and two dimensions by learning suitable convolution matrices, are commonly used in the analysis of physical signals and images. Architecture selection and hyperparameter tuning are central challenges in the implementation of neural networks, and are often performed by an additional machine learning algorithm, for example, using BO (see Section 4.5).

The great strength of neural networks is their flexibility and relatively straightforward implementation, with many openly accessible software platforms available to choose from. However, trained networks are effectively 'black-box' functions, and do not, in their basic form, incorporate uncertainty quantification. As a result, the networks may make over-confident predictions about unseen data while giving no explanation for those predictions, leading to false conclusions. Various methods exist for incorporating uncertainty quantification into neural networks (see, for example, Ref. [103]), such as by including variational layers (discussed above) and training an ensemble of networks on different training data subsets. There are several approaches to try and make neural network models explainable and a review of methods for network interpretation is given, for instance, by Montavon *et al.*[104].

Another strength of neural networks is that the performance of a model on a new task can be improved by leveraging knowledge from a pre-trained model on a related task. This so-called transfer learning is typically used in scenarios where it is difficult or expensive to train a model from scratch on a new task, or when there is a limited amount of training data available. For example, transfer learning has been used to successfully train models for image classification and object detection tasks (see Section 6), using pre-trained models that have been trained on large image datasets such as ImageNet[105]. In the context of laser–plasma physics, transfer learning could be used to improve the performance of a neural network by leveraging knowledge from a pre-trained model that has been trained on data from previous experiments or simulations.

Kirchen *et al.*[29] recently demonstrated the utility of even seemingly small neural networks for modeling laser–plasma accelerator performance. Their multilayer perceptron design is shown in Figure 7 and, as can been seen in Figures 7(b) and 7(c), the network accurately predicts electron beam energy and energy spread. Interestingly, this performance is achieved without using target parameters such as plasma density as an input, thus highlighting the relative importance of laser stabilization in this context. Gonoskov *et al.*[106] trained a neural network to read features in theoretical and experimental spectra from high-order-harmonic generation (HHG) in high-power

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

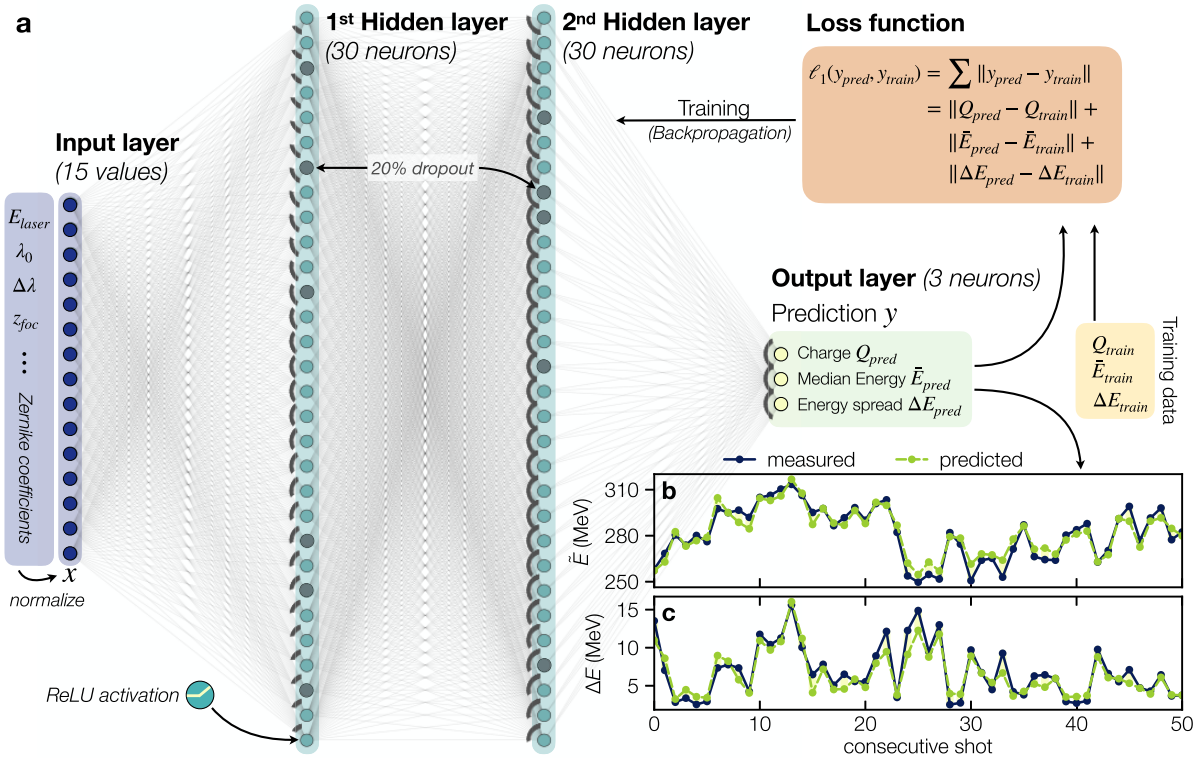

Figure 7. Real-world example of a multilayer perceptron for beam parameter prediction. (a) The network layout[29] consists of 15 input neurons, two hidden layers with 30 neurons and three output neurons (charge, mean energy and energy spread). The input is derived from parasitic laser diagnostics (laser pulse energy E_{laser}, central wavelength λ_0 and spectral bandwidth $\Delta\lambda$, longitudinal focus position z_{foc} and Zernike coefficients of the wavefront). Neurons use a nonlinear ReLU activation and 20% of neurons drop out for regularization during training. The (normalized) predictions are compared to the training data to evaluate the accuracy of the model, in this case using the mean absolute ℓ_1 error as the loss function. In training, the gradient of the loss function is then propagated back through the network to adjust its weights and biases. (b) Measured and predicted median energy ($\bar{E}$) and (c) measured and predicted energy spread (ΔE), both for a series of 50 consecutive shots. Sub-figures (b) and (c) are adapted from Ref. [29].

laser–plasma interactions. Rodimkov *et al.*[107] used a neural network to extract information that was not directly measured in experiments, including the preplasma scale length and the pulse carrier-envelope phase, from the spectrum of extreme ultraviolet (XUV) radiation generated in laser–plasma interactions. Another recent example of deep learning modeling is the work by Djordjević *et al.*[108], where the authors used a multilayer perceptron to model the output of a laser-driven ion accelerator based on training with 1000 simulations. In the work of Watt[109], a strong-field QED model incorporating a trained neural network was used to provide an additional computation package for the Geant4 particle physics platform. Neural networks are also trained to assist hohlraum design for ICF experiments by predicting the time evolution of the radiation temperature, in the recent work by McClarren *et al.*[110]. In the work of Simpson *et al.*[111], a fully connected neural network with three hidden layers is constructed to assist the analysis of an X-ray spectrometer, which measures the X-rays driven by MeV electrons produced from high-power laser–solid interaction. Finally, Streeter *et al.*[112] used convolutional neural networks (CNNs) to predict the electron spectrum produced by a laser wakefield accelerator, taking measurements from secondary laser and plasma diagnostics as the inputs.

2.1.7. Physics-informed machine learning models

The ultimate application of machine learning for modeling physics systems would arguably be to create an 'artificial intelligence physicist', as coined by Wu and Tegmark[113]. One prominent idea at the backbone of how we build physical models is Occam's razor, which states that given multiple hypotheses, the simplest one that is consistent with the data is to be preferred. In addition to this guiding principle, it is furthermore assumed that a physical model can be described using mathematical equations. A program should therefore be able to automatically create such equations, given experimental data. While a competitive artificial intelligence (AI) physicist is still years away,[2] first steps

[2] Recently there has also been astonishing progress in the area of large language models such as generative pre-trained transformer (GPT) models[114]. Very similar to the forecasting networks discussed in Section 2.2.3, these use an attention mechanism to predict the next token (word, etc.) following

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

have been undertaken in this direction. For instance, in 2009 Schmidt and Lipson[116] presented a genetic algorithm approach (Section 4.4) that independently searched and identified governing mathematical representations such as the Hamiltonian from real-life measurements of some mechanical systems. More recently, research has concentrated more on the concept of Occam's razor and the underlying idea that the 'simplest' representation can be seen as the sparsest in some domain. A key contribution was SINDy (sparse identification of nonlinear dynamics), a framework for discovering sparse nonlinear dynamical systems from data[117]. As in CS (Section 3.4), the sparsity constraint was imposed using an ℓ_1-norm regularization, which results in the identification of the fewest terms needed to describe the dynamics.

An important step towards combining physics and machine learning was undertaken in physics-informed neural networks (PINNs)[118]. A PINN is essentially a neural network that uses physics equations, which are often described in form of ordinary differential equations (ODEs) or partial differential equations (PDEs), as a regularization to select a solution that is in agreement with physics. This is achieved by defining a custom loss function that includes a discrepancy term, the residuals of the underlying ODEs or PDEs, in addition to the usual data-based loss components. As such, solutions that obey the selected physics are enforced. In contrast to SINDy, there is no sparsity constraint imposed on the network weights, meaning that the network could still be quite complex. Early examples of PINNs were published by Raissi *et al.*[119–122] and Long *et al.*[123] in 2017–2020. Since then, the architecture has been applied to a wide range of problems in the natural sciences, with a quasi-exponential growth in publications[124]. Applications include, for instance, (low-temperature) plasma physics, where PINNs have been successfully used to solve the Boltzmann equation[125], and quantum physics, where PINNs were used to solve the Schrödinger equation of a quantum harmonic oscillator[126]. The work by Stiller *et al.*[126] uses a scalable neural solver that could possibly also be extended to solve, for example, the Vlasov–Maxwell system governing laser–plasma interaction.

2.2. Time series forecasting

A related problem meriting its own discussion is time series forecasting. While models in the previous section are based on interpolation or regression within given data points, forecasting explicitly deals with the issue of extrapolating parameter values to the future based on prior observations. Most notably this includes modeling of long-term trends (in a laser context often referred to as drifts) or periodic oscillations of parameters and short-term fluctuations referred to as jitter.

If available, models may also use covariates, auxiliary variables that are correlated to the observable, to improve the forecast. These covariates may even extend into the future (seasonal changes being the traditional example in economic forecasting).

In this section we are going to first discuss two common approaches to time series forecasting, autoregressive models and state-space models (SSMs), followed by a discussion of modern techniques based on neural networks. Note that the modeling approaches presented in the preceding section may also be used, to some extent, to extrapolate data. We are not aware of any recent examples on time series forecasting in the context of laser–plasma physics. However, we feel that the topic merits inclusion in this overview as implementations are likely in the near future, for example, for laser stabilization purposes.

2.2.1. Classical models

To model a time series one usually starts with a set of assumptions regarding its structure, that is, the interdependence between values at different times. A simple, approximate assumption is that the observed values in a discrete time series are linearly related. An important, wide-spread class of such models is the so-called autoregressive models, which assert that the next value y_t in a time series is given as a linear function of the p prior values $y_{t-1}, y_{t-2}, \ldots, y_{t-p}$. In addition, each value is assumed to be corrupted by additive white noise $\varepsilon_t \sim \mathcal{N}\left(0, \sigma^2\right)$, representing, for example, measurement errors or inherent statistical fluctuations of the underlying process, which endows the models with stochasticity. In its most common form, the autoregressive model of order p, denoted by AR(p), is defined via the recurrence relation:

$$y_t = \sum_{i=1}^{p} \varphi_i y_{t-i} + \varepsilon_t, \tag{9}$$

where $\varphi_1, \ldots, \varphi_p$ are the model's parameters to be estimated. In this form, the problem of finding the model's parameters is a linear regression problem that can be solved via the least-squares method:

$$\left\{\widehat{\varphi}_1, \ldots, \widehat{\varphi}_p\right\} = \underset{\varphi_1, \ldots, \varphi_p}{\operatorname{argmin}} \left(y_t - \sum_{i=1}^{p} \varphi_i y_{t-i} \right)^2. \tag{10}$$

In another approach we might assume that the next value y_t is instead given by a linear combination of (external) statistical fluctuations, sometimes called shocks, from the past q points in time, again encoded in white noise terms $\varepsilon_t \sim \mathcal{N}\left(0, \sigma^2\right)$. The corresponding model, called the moving average model and denoted by MA(q), is defined as follows:

input provided by the user. It was found that large models ($\sim 10^8 - 10^9$ parameters) become increasingly capable with sufficient training, for example, gaining the ability to do basic math and to write code, including to some claims first hints at artificial general intelligence[115].

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

$$y_t = \mu + \sum_{i=1}^{q} \vartheta_i \varepsilon_{t-i} + \varepsilon_t, \tag{11}$$

where μ is the mean of the time series.

The parameters of the moving average process cannot be inferred by linear methods such as least squares, but rather have to be estimated by means of maximum likelihood methods.

Contrary to the AR(p) model, which is only stationary[3] for certain parameters $\varphi_1, \ldots, \varphi_p$, the MA($q$) model is always (strictly) stationary per definition. Put loosely, stationarity can be understood as the stochastic properties (mean, variance, ...) of the time series being constant over time.

Another distinction between the autoregressive and the moving average model is how far into the future the effects of statistical fluctuations (shocks) are propagated. In the moving average model the white noise terms are only propagated q steps into the future, while in the autoregressive model the effects are propagated infinitely far into the future. This is because the white noise terms are part of the prior values, which themselves are part of the future values in Equation (9).

If the time series in question cannot be explained by the AR(p) or the MA(q) model alone, both models can be combined into what is called an autoregressive-moving-average model, denoted by ARMA (p, q):

$$y_t = \mu + \sum_{i=1}^{p} \varphi_i y_{t-i} + \sum_{j=1}^{q} \vartheta_j \varepsilon_{t-j} + \varepsilon_t. \tag{12}$$

Note that for the special cases of $p = 0$ and $q = 0$, the ARMA (p, q) reduces to the MA(q) and AR(p), respectively.

When fitting autoregressive models, special care should be taken that the time series to be modeled is stationary before fitting, otherwise spurious correlations are introduced. Spurious correlations are apparent correlations between two or more time series that are not causally related, thereby potentially leading to fallacious conclusions as warned of in the well-known adage 'correlation does not imply causation'.

If the time series y is non-stationary in general but stationary with respect to its mean, that is, the variations relative to the mean value are stationary, it might be possible to transform it into a stationary time series. For this we introduce a new series $z_t = \nabla^d y_t$ by differencing the original series d times, where we defined the differencing operator $\nabla y_t \equiv y_t - y_{t-1}$. Applying the differencing operator once ($d = 1$), for example, removes a linear trend from y_t, applying it twice removes a quadratic trend and so forth. If z is stationary after differencing y d-times it can be readily modeled by Equation (12), which leads us to the autoregressive-integrated-moving-average model ARIMA (p, d, q):

$$z_t = \sum_{i=1}^{p} \varphi_i z_{t-i} + \sum_{j=1}^{q} \vartheta_j \varepsilon_{t-j} + \varepsilon_t. \tag{13}$$

Note that in contrast to the ARMA (p, q) model in Equation (12), the time series z_t appearing here is a d-times differenced version of the original series y_t. Further extensions that will only be noted here for completeness are seasonal ARIMA models, called SARIMA models, that allow the modeling of time series that exhibit seasonality (periodicity), and exogeneous ARIMA models, called ARIMAX models, that allow the modeling of time series that are influenced by a separate, external time series. Both extensions can be combined to yield the so-called SARIMAX model, which is general enough to cover a large class of time series problems. However, we are still in any case limited to problems in which the time series is generated by a linear stochastic process. To cover more general nonlinear problems, we introduce SSMs.

2.2.2. *State-space models*

SSMs offer a very general framework to model time series data[127]. In this framework, presuming that the time series is based on some underlying system, it is assumed that there exists a certain true state of the system x_t of which we observe a value y_t subject to measurement noise. The true state x_t, usually inaccessible and hidden to us, and the observed state y_t are modeled by the state equation and the observation equation, respectively. A prominent example of SSMs in machine learning is hidden Markov models (HMMs)[128], in which the hidden state x_t is modeled as a Markov process. In general there are no restrictions on the functional form of the state and observation equations; however, the most common type of SSM is the linear Gaussian SSM, also referred to as the dynamic linear model, in which the state and observation equations are modeled as linear equations and the noise is assumed to be Gaussian. The prototypical example of such linear Gaussian SSMs is the Kalman filter [129], ubiquitous in control theory, robotics and many other fields[130], including for instance adaptive optics[131] and particle accelerator control[132]. The term 'filter' originates from the fact that it filters out noise to estimate the underlying state of a system. The state equations can be written as follows:

$$\begin{aligned} x_t &= A_t x_{t-1} + B_t u_t + C_t \varepsilon_t, \\ y_t &= D_t x_t + E_t \eta_t, \end{aligned} \tag{14}$$

where the system noise ε_t and observation noise η_t are sampled from a normal distribution $\varepsilon_t, \eta_t \sim \mathcal{N}(0, 1)$. Note that the separation of these two noise terms differs from the so-far discussed models. This permits the modeling of

[3] A time series $\{y_t\}$ is said to be strictly stationary, given the joint cumulative distribution $F(y_0, \ldots, y_t)$, if $F(y_0, \ldots, y_t) = F(y_{0+\tau}, \ldots, y_{t+\tau}) \quad \forall \tau \in \mathbb{N}$. Correspondingly, all moments of the joint distribution are invariant under time translation. If this invariance only holds for the first two moments (mean, variance) the time series is said to be (weakly) stationary.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

systems that are driven by noise, while also accounting for observation noise of possibly differing amplitude. Another difference is the addition of a second process u_t, commonly called the *control input* in signal processing, which is a deterministic external process driving the system state. This permits the modeling of systems with known external inputs (e.g., a controller-driven motor) or with known deterministic drivers (e.g., system temperature). Note that we can recover the autoregressive models discussed earlier as a special case of the Kalman filter. For example, the AR(1) process is obtained by setting $A_t = \text{const.} = \varphi_1$, $B_t = 0$, $C_t = \text{const.} = \sigma^2$, $D_t = 1$ and $E_t = 0$. For a more detailed discussion on Kalman filters we refer the interested reader to the literature[133]. Other (nonlinear) ways to perform inference in SSMs exist, for instance using sequential Monte Carlo estimation[134].

2.2.3. *Forecasting networks*

While predictions based on autoregression or SSMs can suffice for many applications, the nonlinear nature of neural networks can be harnessed to model time series with complex, nonlinear dependencies[135,136]. A particularly relevant architecture is the recurrent neural networks (RNNs). These are similar to the 'traditional' neural networks we discussed in Section 2.1.6, but they have a 'memory' that allows them to retain information from previous inputs. This is implemented in a somewhat similar way to the linear recurrence relations discussed in the previous sections, with a key difference being that the state x_t of RNNs is updated according to an arbitrary nonlinear function. The current state x_t is calculated from the previous states $x_{t-1}, x_{t-2}, \ldots$ through the recurrence equation:

$$x_t = f\left(w_{\text{rec}} x_{t-1} + b_{\text{rec}}\right), \tag{15}$$

where f is a nonlinear activation function, w_{rec} is a matrix of recurrent weights and b_{rec} is a bias vector. This equation allows the update to each element of the current state vector, x_t, to be dependent on the whole of the previous state vector, x_{t-1}. The output y_t of the network is calculated from the current state x_t through the output equation:

$$y_t = f\left(w_{\text{out}} x_t + b_{\text{out}}\right), \tag{16}$$

where w_{out} contains the output weights and b_{out} is another bias vector. Thus, the entire network state is updated from the previous state through a recurrent equation, and the current output is calculated from the current state through an output equation. The network state therefore contains all previous information about the time series. However, as the network state is updated by a simple multiplication of the weights w_{rec} with the previous state x_{t-1}, the so-called vanishing gradient problem may occur[137,138]. This problem is solved in a seminal work by Hochreiter and Schmidhuber[139], who introduced a special type of RNN, the long short-term memory (LSTM) network. In contrast to the simple update equation of RNNs, LSTMs use a special type of memory cell that can learn long-term dependencies. This includes a so-called forget gate f_t to update the previous state x_{t-1} to the current state x_t:

$$x_t = f_t \odot x_{t-1} + i_t \odot h_t, \tag{17}$$

where $\odot$ denotes the element-wise Hadamard product, $[A \odot B]_{ij} = A_{ij} B_{ij}$. In addition, the LSTM uses two more gates, the input gate i_t and output gate o_t, where the latter is used to calculate the hidden state h_t from the previous hidden state via $h_t = o_t \odot h_{t-1}$. The three gates f_t, i_t and o_t are calculated from the current input x_t, the previous hidden state h_{t-1} and the previous gate states f_{t-1}, i_{t-1} and o_{t-1} using individually set weights and biases. The gates determine how much of the previous state x_{t-1} and the current hidden state h_t are used to calculate the current state x_t. The output of the LSTM network is then calculated from the current state x_t through the same output equation, Equation (16), as used in the RNN. Despite it being introduced in the early 1990s, the LSTM architecture remains one of the most popular network architectures for predictive tasks to date.

A more recently introduced type of neural network that can learn to interpret and generate sequences of data is the transformer. Transformers are similar to RNNs, but instead of processing the time series in a sequential manner, they use a so-called attention mechanism[140] to capture dependencies between all elements of the time series simultaneously and, thus, focus on specific parts of an input sequence. Assuming again a time series $x_t, x_{t-1}, \ldots$, a transformer maps each point x_t to a representation h_t using a linear layer, for example, $h_t = w x_t + b$. The transformer network then calculates a new representation for each point x_t using the attention mechanism:

$$\tilde{h}_t = \sum \alpha_{ti} h_i, \tag{18}$$

where the attention weights α_{ti} are calculated using the point representations h_i and the previous representation $\tilde{h}_t$. The attention weights α_{ti} are used to calculate the new representation $\tilde{h}_t$ of each point x_t from all previous representations $\tilde{h}_i$, $i < t$, of the previous point x_i, $i < t$. The new representations are then used to calculate the output y_t of the network as in Equation (16). Thus, the attention mechanism of a transformer network can be interpreted as a nonlinear function that updates the representation of each point x_t from all previous representations of the previous points x_i, $i < t$. It can be applied multiple times and enables transformers to learn complex patterns in data, outperforming RNNs on a variety of tasks, such as machine translation and language modeling. It has also been shown that transformers are more efficient than RNNs, meaning they can be trained on larger datasets in less time. One recent example of a transformer

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

developed for time series prediction is the Temporal Fusion Transformer[141]. Beyond RNNs, LSTMs and transformers, there exist many other network architectures that can be used for forecasting, for example, fully convolutional networks (FCNs), as discussed by Wang *et al.*[142].

For longer-term forecasting, predictive networks are usually employed iteratively, meaning that a single-step forecast uses only real data, while a two-step forecast will use the historical data and the most recent prediction value, and so forth. In the context of laser–plasma physics and experiments, predictive neural networks could be used to model the time series data from diagnostic measurements in order to make predictions about the future performance, for example, for predictive steering of laser or particle beams.

2.3. Prediction and feedback

Both (surrogate) models and forecasts can be used to make predictions about the state of a system at a so-far unknown position, in parameter space or time, respectively. This type of operation is sometimes referred to as open-loop prediction. Closed-loop prediction and feedback, on the other hand, use predictions and compare them to the actual system state, continuously updating and improving the surrogate model of the system. This is particularly relevant for dynamic systems, where parameters change over time. A complete discussion on how to implement a closed-loop system using machine learning goes beyond the scope of this review, as it would also require an extensive discussion of control systems and so forth. The following discussion will thus be restricted to a brief outline of a few relevant concepts.

A feedback loop is most generally a system where a part of the output serves as input to the system itself, and we have already discussed some models with feedback in the context of forecasting (Section 2.2). Another well-known engineering implementation of closed-loop operation is the proportional–integral–derivative (PID) controller. A PID controller adjusts the process variable (PV) with a proportional, integral and derivative response to the difference between a measured PV and a desired set point. Here the proportional term serves to increase the response time of the system to changes in the set point. The integral term is used to eliminate the residual steady-state error that occurs if the PV is not equal to the set point. The derivative term improves the stability of the control, reducing the tendency of the PID output to overshoot the set point.

In the context of laser–plasma physics, the implementation of the feedback loop would most likely look slightly different. One possible implementation would be that a model is used to predict the output of the system at a given set of parameters, which is then compared to the actual output and used to update the model again. In order to keep improving the model, it is important that new data should be acquired in regions of parameter space that deviate from the previously acquired data. In other words, it is important to explore parameter space in addition to exploiting the knowledge from existing data points. This can be done through random sampling or through so-called BO, as we will discuss in Section 4.

Such a model that is continuously updated with new data is sometimes referred to as a dynamic model. There are two main approaches to updating such models.

- Firstly, completely re-training the model with all available data, including the newly acquired data points. This results in an increasingly large training dataset and training time.
- The second method is to update the model by adding new points to the training set and then re-training the existing model on these points in a process known as incremental learning. This method is thus much faster and uses less memory than a full re-training.

However, not all techniques we discussed to construct models are compatible with both training methods. For instance, GP regression can historically only be trained on full datasets, hindering its applicability in settings requiring (near) real-time updates. However, recent work on regression in a streaming setting might alleviate this problem[143]. Learning dynamic models is further complicated by the fact that systems may change over the course of time during which training data are acquired, a problem referred to as concept drift[144].

Many laboratories are currently stepping up their efforts to integrate prediction and feedback systems into their lasers and experiments[145,146]. One of the first groups to extensively make use of these techniques was the team around A. Maier at DESY's LUX facility[26]. Among the most comprehensive proposals are plans recently presented by Ma *et al.* from NIF, proposing an extensive integration of feedback systems for high-energy-density (HED) experiments[147]. The system, referred to as a 'full integrated high-repetition rate HED laser–plasma experiment', consists of multiple linked feedback loops. These are hierarchically organized, starting from a 'laser loop', over an 'experiment loop' to a 'modeling & simulation loop' at its highest level.

3. Inverse problems

In the previous section we looked at the forward problem of modeling the black-box function $f(x) = y$. In many scientific and engineering applications, it is necessary to solve the inverse problem, namely to determine x given y and f (or an approximation $f^* \approx f$). Inverse problems are extremely common in experimental physics, as they essentially describe the

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

measurement process and subsequent retrieval of underlying properties in a physics experiment.

In many cases the problem takes the form of a discrete linear system:

$$Ax = y, \tag{19}$$

where y is the known observation and A describes the measurement process acting on the unknown quantity x to be estimated. In most cases we can assume that the system behaves linearly with respect to the input parameters x and, hence, A can be written as a single sensing matrix. However, more generally, A can be thought of as an operator, which maps the input vector x to the observation vector y.[4]

A classical example of an inverse problem is CT, whose goal is to reconstruct an object from a limited set of projections at different angles. Other examples are wavefront sensing in optics, the 'FROG' algorithm for ultra-fast pulse measurements, the 'unfolding' of X-ray spectra or, in the context of particle accelerators, the estimation of particle distributions from different measurements of a beam.

3.1. Least-squares solution

A common approach to the problem described by Equation (19) is to use the least-squares approach. Here, the problem is reformulated as minimizing the quadratic, positive error between the observation and the estimate:

$$\widehat{x}_{\mathrm{LS}} = \underset{x}{\operatorname{argmin}} \left\{ \|Ax - y\|^2 \right\}. \tag{20}$$

If A is square and non-singular, the solution to this equation is obtained via the inverse, $A^{-1}y$. For non-square matrices, the pseudo-inverse can be used, which can for instance be computed via singular value decomposition (SVD)[148]. Alternatively, a multitude of iterative optimization methods can be employed, for example, iterative shrinkage-thresholding algorithms[149,150] or gradient descent, to name a few. For more details on the solution to this optimization problem, the reader is referred to Ref. [151].

The approach given by Equation (20) is widely used in overdetermined problems, where the regression between data points with redundant information can help to reduce the influence of measurement noise. Prominent application areas are, for instance, wavefront measurements and adaptive optics[152].

3.2. Statistical inference

As for predictive models (Section 2.1.3), one can also approach inverse problems via probabilistic methods, for example, if the underlying model $f(x)$ is stochastic in nature or measurements are corrupted by noise. One popular approach is to use MLE[153]. As we have seen before, MLE consists of finding the value that maximizes the likelihood (see Equation (1)). To this end, one often uses Markov chain Monte Carlo (MCMC) methods[154] and/or gradient descent algorithms. Alternatively, expectation-maximization (EM) is a popular method to compute the maximum likelihood estimate[155] and is, for instance, often used in statistical iterative tomography (SIR)[156]. EM alternates between an estimate step, where one computes the expectation of the likelihood function, and a maximization step, where one obtains a new estimate that maximizes the posterior distribution. By sequentially repeating the two steps, the estimate converges to the maximum likelihood estimate.

Both the least-squares and MLE approaches suffer from the issue that the result is a point estimate. Thus, if the underlying problem is ill-posed or underdetermined, this estimate is often not unique or representative, or the least-squares solution is prone to artifacts resulting from small fluctuations in the observation. Consequently, it is often desirable to obtain an estimate of the entire solution space, that is, a probability distribution of the unknown parameter x. The latter allows one to not only compute the estimate $\widehat{x}$, but also to obtain a measure of the estimation uncertainty $\sigma_{\widehat{x}}$.

To this end, one can reformulate the inverse problem as a Bayesian inference problem and get both an expectation value and an uncertainty from the posterior probability $p(y|x)$. As we have seen in Section 2.1.3, calculating the posterior requires knowledge of the likelihood $p(x|y)$[153]. However, in some settings the forward problem $f(x)$ can be very expensive to evaluate, which is for instance the case for accurate simulations of laser–plasma physics. In this case the likelihood function $p(x|y)$ becomes intractable because it requires the computation of $f(x)$ for many values of x. Historically, solutions to this problem have been referred to as methods of likelihood-free inference, but more recently the term simulation-based inference is being used increasingly[157]. One of the most popular methods in this regard is approximate Bayesian computation (ABC)[158],[5] which addresses the issue by employing a reject–accept scheme to estimate the posterior distribution $p(y|x)$ without needing to compute the likelihood $p(x|y)$ for any value of x. Instead, ABC first randomly selects samples x' from a predefined prior distribution, which is usually done using an MCMC[159] algorithm or, in more recent work, via BO[160]. Defining good priors might require some expert knowledge about the system. Subsequently, one simulates the observation $y' = f(x')$ corresponding to each sample. Finally, the ABC algorithm accepts only those samples that match the observation within a set tolerance ϵ, thereby essentially approximating the likelihood function with a rejection

[4] The sensing matrix/operator is known by different names, for example, the instrument response, the system matrix, the response matrix, the transfer function or, most generally, the forward operator.

[5] While both rely on Bayesian statistics and thus have some conceptual overlap, approximate Bayesian computation should not be confused with Bayesian optimization, discussed in Section 4.5.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

probability $\rho(y, y')$. This yields a reduced set of samples, whose distribution approximates the posterior distribution. The tolerance ϵ determines the accuracy of the estimate; the smaller ϵ, the more accurate the prediction, but due to the higher rate of rejection one also requires more samples of the forward process. This can make ABC prohibitively expensive, especially for high-dimensional problems. Approximate frequentist computation[161] is a conceptually similar approach that approximates the likelihood function instead of the posterior. Information on this method and other simulation-based inference techniques, including their recent development in the context of deep learning, for example, for density estimation in high dimensions, can be found, for instance, in the recent overview paper by Cranmer *et al.*[157].

3.3. Regularization

One way to improve the estimate of the inverse in the case of ill-posed or underdetermined problems is to use regularization methods. Regularization works by further conditioning the solution space, thus replacing the ill-conditioned problem with an approximate, well-conditioned one. In variational regularization methods this is done by simultaneously solving a second minimization problem that incorporates a desirable property of the solution, for example, minimization of the total variation to remove noise.[6] Such regularization problems hence take the following form:

$$\widehat{x}_{\mathrm{REG}} = \underset{x}{\operatorname{argmin}} \left\{ \|Ax - y\|^2 + \lambda \mathcal{R}(x) \right\}, \tag{21}$$

where λ is a hyperparameter controlling the impact of the regularization and $\mathcal{R}$ is the regularization criterion, for example, a matrix description of the first derivative. The effect of the regularization can be further adjusted by the norm that is used to calculate the residual term, for example, using the absolute difference, the squared difference or a mix of both, such as the Huber norm.[7] The more complex optimization problem in Equation (21) itself is typically solved using iterative optimization algorithms, as already mentioned in the previous section. As will be discussed in Section 3.6, there have been recent developments in data-driven approaches to learning the correct form of $\mathcal{R}(x)$, by representing it with a neural network.

A recent demonstration in the context of LPA is the work by Döpp *et al.*[162,163], who presented results on tomographic reconstruction using betatron X-ray images of a human bone sample; see Figure 8. The 180 projection images were acquired within 3 minutes and an iterative reconstruction of the object's attenuation coefficients was performed. For regularization, a variation of Equation (21) was used that used the Huber norm and included a weighting factor in the data term, whose objective was to adjust the weight of poorly illuminated detector pixels.

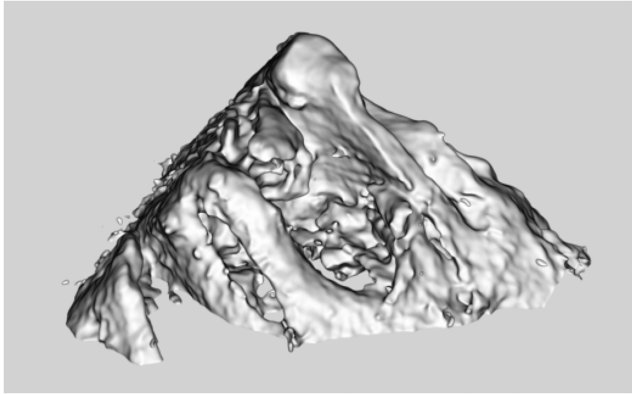

Figure 8. Tomography of a human bone sample using a laser-driven betatron X-ray source. Reconstructed from 180 projections using statistical iterative reconstruction. Based on the data presented by Döpp *et al.*[162].

3.4. Compressed sensing

Compressed sensing (CS)[164–166] is a relatively new research field that has attracted significant interest in recent years, since it efficiently deals with the problem of reconstructing a signal from a small number of measurements. The mathematical theory of CS has proven that it is possible to reliably reconstruct a complex signal from a small number of measurements, even below the Shannon–Nyquist sampling rate, as long as two conditions are satisfied. Firstly, the signal must be 'sparse' in some other representation (i.e., it must contain few non-zero elements). In this case we can replace the dense unknown variable x with its sparse counterpart $\tilde{x}$ by using the transformation matrix Ψ corresponding to a different representation, that is, $x = \Psi\tilde{x}$. Here, Ψ could for instance be the wavelet transformation. The second condition concerns the way the measurements are taken (hence compressed sensing): the sensing matrix A must be incoherent with respect to the sparse basis Ψ, which ensures that the sparse representation of the signal is fully sampled.

At its core, CS is closely related to the concepts of regularization discussed in the previous section. In order to reconstruct the signal from the measurements, the ideal regularization, $\mathcal{R}(\tilde{x})$, is that which sums the number of non-zero components of $\tilde{x}$ and thus promotes sparsity when minimized[167]. However, it has been shown that using the

[6] Another class of regularization methods is based on smoothing in some sense. For instance, filtered back projection can be understood as a smoothing of the projection line integrals by means of a convolution with a filter function.

[7] It should be noted that a very similar problem formulation can also be found in the context of training neural networks, where regularization terms are often added to the cost function.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

vastly more computationally efficient ℓ_1 norm leads to the same results on many occasions[168], and thus the CS reconstruction problem can be written as follows:

$$\widehat{x}_{\mathrm{CS}} = \underset{\tilde{x}}{\operatorname{argmin}} \left\{ \|A\Psi\tilde{x} - y\|^2 + \lambda \|\tilde{x}\|_1 \right\}. \tag{22}$$

It should be noted that CS is not the first method to achieve sub-Nyquist sampling in a certain domain; see, for example, band-limited sampling[169]. The strength of the formalism is rather that it is very flexible, because it only requires the coefficients of a signal to be sparse, without exactly knowing beforehand which ones are non-zero.

It should be noted that in the most general case, the basis on which the signal is sparse, Ψ, is unknown. Nonetheless, the incoherence with the sampling basis can be satisfied by sampling randomly. To reconstruct the signal from such measurements, one returns to solving Equation (21). This can still be considered as CS, due to the fact that the nature of the sensing process is designed to exploit the sparsity of the signal. Therefore, CS can be used alongside deep-learning-based approaches to solve Equation (21), which will be discussed in the subsequent sections.

CS has been used in a number of fields related to laser–plasma physics, for example ultra-fast polarimetric imaging [170] and ICF radiation symmetry analysis[171]. There exist also several examples from the context of tomographic reconstruction[172,173]. Such enhanced reconstruction algorithms can reduce the number of projections beyond what is usually possible with regression techniques. For instance, in the work by Ma *et al.*[173] a test object was illuminated using a Compton X-ray source and a compressive tomographic reconstruction algorithm[174] was used to reconstruct the sample's well-defined inner structure from only 31 projections.

3.5. *End-to-end deep learning methods*

In recent years, there has been growing interest in applying deep learning to inverse problems[175]. In general, these can be categorized into two types of approaches, namely those that aim to entirely solve the inverse problem end-to-end using a neural network and hybrid approaches that replace a subpart of the solution with a network (see the next section). Denoting the artificial neural network (ANN) as $\mathcal{A}$, the estimate $\widehat{x}_{\mathcal{A}}$ from the end-to-end network could be simply written as follows:

$$\widehat{x}_{\mathcal{A}} = \mathcal{A}y. \tag{23}$$

ANNs can thus be interpreted as a nonlinear extension of linear algebra methods, such as SVD[176]. As such, for well-posed problems, the ANN is acting similarly to the least-squares method and, if provided with non-biased training data, directly approximates the (pseudo-)inverse.

However, using neural networks for ill-posed problems is more difficult, as training tends to become unstable when the networks need to generate data, that is, the layer containing the desired output x is larger than the input y. Fortunately, several network architectures have been developed that perform very well at these tasks, such as generative adversarial networks (GANs)[177]. GANs are two-part neural networks, one of which is trying to generate data that resemble the input (the generator), while the other is trying to distinguish between the generated and real data (the discriminator). As the two networks compete against each other they improve their respective skills, and the generator will eventually be able to create high-quality data. The generator is then used to estimate the solution of the inverse problem. Training GANs can still be challenging, with common issues such as mode collapse[178]. Autoencoders (AEs) are a simpler architecture that can perform similar tasks with relevance to inverse problems[179]. U-Nets[180] are a popular architecture that draws inspiration from AEs and have proven to be very powerful for inverse problems, especially in imaging. They essentially combine features of AEs with FCNs, in particular ResNets (see also Section 6.1.2): similar to an AE, U-Nets consist of an encoder part, followed by a decoder part that usually mirrors the encoder. At the same time, the layers in the network are skip-connected, meaning that the output from the previous layer is concatenated with the output of the corresponding layer in the encoder part of the network (see Figure 6). This allows the network to retain information about the input data even when it is transformed to a lower dimensional space.

One sub-class of ANNs that has gained considerable recent interest in the context of inverse problems is the invertible neural network (INN)[181]; see sketch in Figure 9. Mathematically, an INN is a bijective function between the input and output of the network, meaning that it can be exactly inverted. Because of this, an INN trained to approximate the forward function A will implicitly also learn an estimate for its inverse. In order to be applied to inverse problems, both forward and inverse mapping should be efficiently computable. In ill-posed problems there is an inherent information loss present in the forward process, which is either counteracted by the introduction of additional latent output variables z, which capture the information about x that is missing in y, or by adding a zero-padding of corresponding size. The architecture and training of INNs are inspired by recent advances in normalizing flows[182]. The name stems from the fact that the mapping learned by the network is composed of a series of invertible transformations, called coupling blocks[183], which 'normalize' the data, meaning that they move the data closer to a standard normal distribution $\mathcal{N}(0,1)$. The choice of coupling block

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

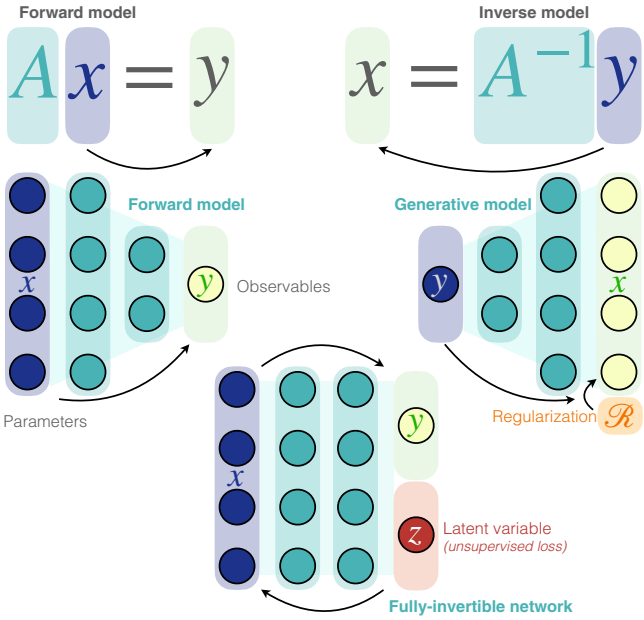

Figure 9. Deep-learning for inverse problems. Sketch explaining the relation among predictive models, inverse models and fully invertible models.

basically restricts the Jacobian of the transformation between the standard normal latent variable and the output. INNs can be trained bi-directionally, meaning that the loss function is optimized using both the loss of the forward pass and the inverse pass. In addition, an unsupervised loss such as maximum mean discrepancy[184] or negative log-likelihood[185] can be used to encourage the latent variable to be as close to a standard normal variable as possible.

The loss function of end-to-end networks can also be modified such that a classical forward model is used in the loss function. Such PINNs (see Section 2.1.7) respect the physical constraints of the problem, which can lead to more accurate and physically plausible solutions for the inverse problem.

An early example from the context of ultra-fast laser diagnostics was published by Krumbügel *et al.* in 1996[186], where the authors tested a neural network for retrieval of laser pulse profiles from FROG traces. At the time, numerical capabilities were much more limited than today and the FROG trace containing some 64×64 data points was at the time considered as 'far too much for a neural net'. Because of this, the data had to be compressed and only a polynomial dependence was trained, limiting reconstruction performance. The vast increase in computing power over the past decades has largely alleviated this issue and the problem was revisited by Zahavy *et al.* in 2018[187]. Their 'DeepFROG' network is based on a CNN architecture and, now using the entire 2D FROG trace as input, achieves a similar performance as iterative methods.

Another example for an end-to-end learning of a well-conditioned problem was published by Simpson *et al.*[111], who trained a fully connected three-layer network on a large set (47,845 samples) of synthetic spectrometer data to retrieve key experimental metrics, such as particle temperature.

U-Nets have been used for various inverse problems, including, for example, the reconstruction of wavefronts from Shack–Hartmann sensor images as presented by Hu *et al.*[188] (see Figure 10).

An example for the use of INNs in the context of LPA was recently published by Bethke *et al.*[189]. In their work they used an INN called iLWFA to learn a forward surrogate model (from simulation parameters to the beam energy spectrum) and then used the bijective property of the INN to calculate the inverse, that is, deduce the simulation parameters from a spectrum.

3.6. Hybrid methods

In contrast to end-to-end approaches, there exist a variety of hybrid schemes that employ neural networks to solve part of the inverse problem. A collection of such methods focuses on splitting Equation (21) into two subproblems, separating the quadratic loss from the regularization term. The former is easily minimized as a least-squares problem (discussed in Section 3.1) and the optimum form of regularization can then be learned by a neural network. Crucially, this network can be smaller and more parameter-efficient than in end-to-end approaches, as it has a simpler task and less abstraction to perform. Furthermore, the direct relation that this network has the regularization $\mathcal{R}(x)$ allows one to pick a network structure to exploit prior knowledge about the data.

There are multiple methods available to perform such a separation, for example half quadratic splitting (HQS) or the alternating direction method of multipliers (ADMM). HQS is the simplest method and involves substituting an auxiliary variable, p, for x in the regularization term and then separating Equation (21) into two subproblems, which are solved iteratively. The approximate equality of p and x is ensured by the introduction of a further quadratic loss term into each of the subproblems: $\beta\|x-p\|^2$. If the same neural network (with the same set of weights) is used to represent the regularization subproblem in each iteration, the method is referred to as plug-and-play, but if each iteration uses a separate network, the method is deep unrolling[190]. Such methods have achieved unprecedented accuracy whilst reducing computational cost.

It is worth noting that there exist other similar methods – for instance, neural networks can be used to learn an appropriate regression function $\mathcal{R}(x)$ in Equation (21), as for instance done in network Tikhonov (NETT)[191].

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

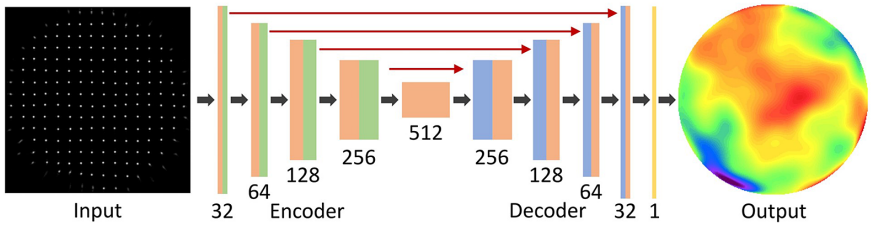

Figure 10. Application of the end-to-end reconstruction of a wavefront using a convolutional U-Net architecture[180]. The spot patterns from a Shack–Hartmann sensor are fed into the network, yielding a high-fidelity prediction. Adapted from Ref. [188].

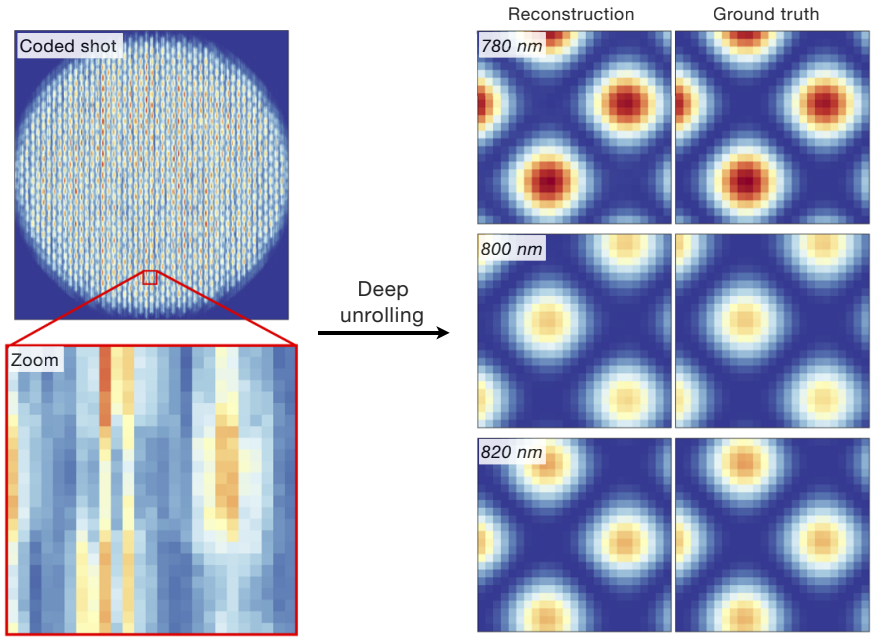

Figure 11. Deep unrolling for hyperspectral imaging. The left-hand side displays an example of the coded shot, that is, a spatial-spectral interferogram hypercube randomly sampled onto a 2D sensor. The bottom left shows a magnification of a selected section. On the right-hand side is the corresponding reconstructed spectrally resolved hypercube. Adapted from Ref. [192].

Howard *et al.*[192] recently presented an implementation of CS using deep unrolling for single-shot spatio-temporal characterization of laser pulses. Such a characterization is a typical example of an underdetermined problem, where one wishes to capture 3D information (across the pulse's spatial and temporal domains) on a 2D sensor. Whilst previous methods mostly resorted to scanning, resulting in long characterization times and blindness to shot-to-shot fluctuations, this work presented a single-shot approach, which has numerable benefits. The authors' implementation is based on a lateral shearing interferometer to encode the spatial wavefront in an interferogram for each spectral channel of the pulse, creating an interferogram cube. An optical system called CASSI[193] was then used to randomly sample this 3D data onto a 2D sensor resulting in a 'coded shot', thereby fulfilling the conditions of CS. For the reconstruction of the interferogram cube, the HQS method was utilized, and the regularization term was represented by a network with 3D convolutional layers that can capture correlations between the spectral and spatial domains. An example for a successful reconstruction is shown in Figure 11.

4. Optimization

One of the most common problems in applied laser–plasma physics experiments, in particular LPA, is the optimization of the performance through manipulation of the machine controls. Here the goal is to minimize or maximize an objective function, a metric of the system performance according to some pre-defined criteria (see Section 4.1.1). A simple case of this would be to optimize the total charge of the accelerated electron beam, but in principle, any measurable

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

Table 1. Summary of a few representative papers on machine-learning-aided optimization in the context of laser–plasma acceleration and high-power laser experiments.

Author, year	Laser type	Optimization method(s)	Free parameters	Optimization goals
He *et al.*, 2015[194]	800 nm Ti:Sa, 15 mJ, 35 fs, 0.5 kHz	Genetic algorithm	Deformable mirror (37 actuator voltages)	Electron angular profile, energy distribution & transverse emittance, optical pulse compression
Dann *et al.*, 2019[195]	800 nm Ti:Sa, 450 mJ, 40 fs, 5 Hz	Genetic & Nelder–Mead algorithms	Deformable mirror or acousto-optic programmable dispersive filter	Electron beam charge, total charge within energy range, electron beam divergence
Shalloo *et al.*, 2020[196]	800 nm Ti:Sa, 0.245 J, 45 fs (bandwidth limit), 1 Hz	Bayesian optimization	Gas cell flow rate & length, laser dispersion ($\partial^2_\omega\phi$, $\partial^3_\omega\phi$, $\partial^4_\omega\phi$), focus position	Total electron beam energy, electron charge within acceptance angle, betatron X-ray counts
Jalas *et al.*, 2021[197]	800 nm Ti:Sa, 2.6 J, 39 fs, 1 Hz	Bayesian optimization	Gas cell flow rates (H_2 front and back, N_2); focus position and laser energy	Spectral charge density

beam quantity or combination of quantities can be used to create the objective function.

There are many different general techniques for tackling optimization problems, and their suitability depends on the type of problem. Single shots in an LPA experiment (or a single run of numerical simulation) can be considered the evaluation of an unknown function that one wishes to optimize. The form of this function is not typically known, due to the lack of a full theoretical description of the experiment, and therefore the Jacobian of this function is also unknown. The input to this function is typically highly dimensional, due to the large number of machine control parameters that affect the output, and these input parameters are coupled in a complex and often nonlinear manner. Evaluation of this unknown function is also relatively costly, meaning that optimization should be as efficient as possible to minimize the required beam time or computational resources. In addition, the unknown function has some stochasticity, due to the statistical noise in the measurement and also due to the natural variation of unmeasured input parameters, which nonetheless may contribute significantly to variations in the output. Finally, there are usually constraints placed upon the input parameters due to the operation range of physical devices and machine safety requirements. These constraints might also be non-trivial due to coupling between different input parameters, and may also depend on system outputs (e.g., to avoid beam loss in an accelerator).

Due to all these considerations, not all optimization algorithms are suitable and only a few different types have been explored in published work; see Table 1 for a selection of representative papers. The following sections will focus on these methods. The reader is referred to dedicated reviews, for example, the comprehensive work by Nocedal and Wright[151] on numerical optimization algorithms, for further reading.

4.1. General concepts

4.1.1. Objective functions

Most optimization algorithms are based on maximizing or minimizing the value of the objective function, which has to be constructed in a way that it represents the actual, user-defined objective of the optimization problem. Typically, the objective function produces a single value, where higher (or lower) values represent a more optimized state.[8] The optimization problem is then a case of finding the parameters that maximize (or minimize) the objective function.

When the objective is to construct a model, the objective function encodes some measure of similarity between the model and what it is supposed to represent, for example, a measure of how well the model fits some training data. For example, deep learning algorithms minimize an objective function that encodes the difference between desired output values for a given input and actual results produced by the algorithm. A common, basic metric for this is the MSE cost function, in which the difference between predicted and actual values is squared. We already mentioned this metric at several points of this review, for example, in Section 3.1. The MSE objective function belongs to a class of distance metrics, the ℓ-norms, which are defined by the following:

$$\ell_p(x,y) = \left(\sum \| x-y\|^p\right)^{1/p}, \tag{24}$$

where x and y are vectors, and we use the notation $\ell_2 = \text{MSE}$. One can also use other ℓ-norms, for example, ℓ_1, which is

[8] Depending on the context of optimization problems, this function is referred to using many different names, for example, merit function (using a figure of merit), fitness function (in the context of evolutionary algorithms), cost or loss function (in deep learning) or reward function (reinforcement learning). Some of these are associated with either maximization (reward, fitness, …) and others with minimization (cost, loss). Here, we will use the general term objective function as used in optimization theory.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

more robust to outliers than ℓ_2 since it penalizes large deviations less severely. Note that the number of non-zero values is sometimes referred to as the 'ℓ_0' norm (see Section 3.4), even though it does not follow from Equation (24).

Another popular similarity measure is the Kullback–Leibler (KL) divergence, sometimes called relative entropy, which is used to find the closest probability distribution for a given model. It is defined as follows:

$$\mathrm{KL}\,(p|q) = \sum p(x) \log \frac{p(x)}{q(x)}, \tag{25}$$

where $p(x)$ is the probability of observing the value x according to the model, and $q(x)$ is the actual probability of observing x. The KL divergence is minimized when the model prediction $p(x)$ is as close to the actual distribution $q(x)$ as possible. It is a relative measure, that is, it is only defined for pairs of probability distributions. The KL divergence is closely related to the cross-entropy cost function:

$$\mathrm{CE}\,(p,q) = -\sum p(x) \log q(x) = \mathrm{KL}\,(p|q) - H(p), \tag{26}$$

where $p(x)$ and $q(x)$ are probability distributions, and $H(p) = -\sum p(x) \log p(x)$ is the Shannon entropy of the distribution $p(x)$.

Other objective functions may rely on the maximization or minimization of certain parameters, for example, the beam energy or energy-bandwidth of particle beams produced by a laser–plasma accelerator. Both of these are examples of what is sometimes referred to as summary statistics, as they condense information from more complex distributions, in this case the electron energy distribution. While simple at the first glance, these objectives need to be properly defined and there are often different ways to do so[198]. In the example above, energy and bandwidth are examples for the central tendency and the statistical dispersion of the energy distribution, respectively. These can be measured using different metrics, such as the weighted arithmetic or truncated mean, the median, mode, percentiles, for the former; and full width at half maximum, median absolute deviation, standard deviation, maximum deviation, for the latter. Each of these seemingly similar measures emphasizes different features of the distribution they are calculated from, which can affect the outcome of optimization tasks. Sometimes one might also want to include higher-order momenta as objectives, such as the skewness, or use integrals, for example, the total beam charge.

4.1.2. *Pareto optimization*

In practice, optimization problems often constitute multiple, sometimes competing objectives g_i. As the objective function should only yield a single scalar value, one has to condense these objectives in a process known as scalarization. Scalarization can, for instance, take the form of a weighted product $g = \prod g_i^{\alpha_i}$ or sum $g = \sum \alpha_i g_i$ of the individual objectives g_i with the hyperparameters α_i describing the weight. Another common scalarization technique is ϵ-constraint scalarization, where one seeks to reformulate the problem of optimizing multiple objectives into a problem of single-objective optimization conditioned on constraints. In this method the goal is to optimize one of the g_i given some bounds on the other objectives. All of these techniques introduce some explicit bias in the optimization, which may not necessarily represent the desired outcome. Because of this, the hyperparameters of the scalarization may have to be optimized themselves by running optimizations several times.

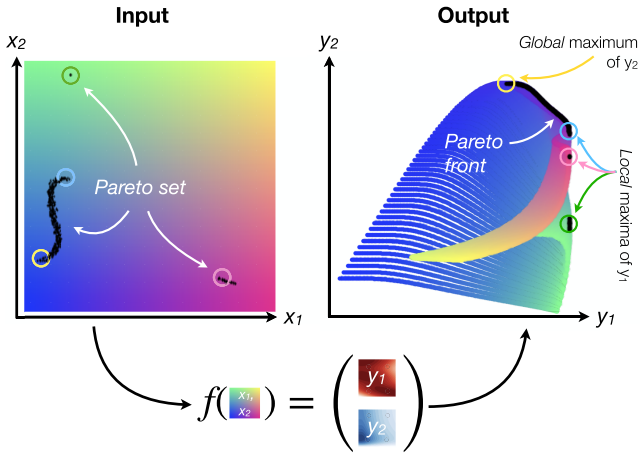

Figure 12. Pareto front. Illustration of how a multi-objective function $f(x) = y$ acts on a 2D input space $x = (x_1, x_2)$ and transforms it to an objective space $y = (y_1, y_2)$ on the right. The entirety of possible input positions is uniquely color-coded on the left and the resulting position in the objective space is shown in the same color on the right. The Pareto-optimal solutions form the Pareto front, indicated on the right, whereas the corresponding set of coordinates in the input space is called the Pareto set. Note that both the Pareto front and Pareto set may be continuously defined locally, but can also contain discontinuities when local maxima become involved. Adapted from Ref. [199].

A more general approach is Pareto optimization, where the entire vector of individual objectives $g = (g_1, \ldots, g_N)$ is optimized. To do so, instead of optimizing individual objectives, it is based on the concept of dominance. A solution is said to dominate other solutions if it is both at least as good on all objectives and strictly better than the other solutions on at least one objective. Pareto optimization uses implicit scalarization by building a set of non-dominated solutions, called the Pareto front, and maximizing the diversity of solutions within this set. The latter can, for instance, be quantified by the hypervolume of the set or the spread of solutions along each individual objective. As it works on the solution for all objectives at once, Pareto optimization is commonly referred to as multi-objective optimization. An illustration of both the Pareto front and set is shown in Figure 12, where a multi-objective function f 'morphs'

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

the input space into the objective space. In this example f is a modified version of the Branin–Currin function[200,201], exhibiting a single, global maximum in y_2 but multiple local maxima in y_1. The individual 2D outputs $y_1 = f_1(x_1, x_2)$, with f_1 being the Branin function, and $y_2 = f_2(x_1, x_2)$, with f_2 being the Currin function, are depicted with red and blue colormaps at the bottom.

4.2. Grid search and random search

Once an objective is defined, we can try to optimize it. One of the simplest methods to do so is a grid search, where the input parameter space is explored by taking measurements in regularly spaced intervals. This technique is particularly simple to implement, especially in experiments, and therefore remains very popular in the setting of experimental laser physics. However, the method severely suffers from the 'curse of dimensionality', meaning that the number of measurements increases exponentially with the number of dimensions considered. In practice, the parameter space therefore has to be low-dimensional (1D, 2D or 3D at most) and it is applied to the optimization of selected parameters that appear to influence the outcome the most. One issue with grid scans is that they can lead to aliasing, that is, high-frequency information can be missed due to the discrete grid with a fixed sampling frequency.

A popular variation, in particular in laser–plasma experiments, is the use of sequential 1D line searches. Here, one identifies the optimum in one dimension, then performs a scan of another parameter and moves to its optimum, and so forth. This method can converge much faster to an optimum, but is only applicable in settings with a single, global optimum.

Random search is a related method where the sampling of the input parameter space is random instead of regular. This method can be more efficient than grid search, especially if the system involves coupled parameters and has a lower effective dimensionality[202]. It is therefore often used in optimization problems with a large number of free parameters. However, purely random sampling also has drawbacks. For instance, it has a tendency to cluster, that is, to sample points very close to others, while leaving some areas unexplored. This behavior is undesirable for signals without high-frequency components and instead one would rather opt for a sampling that explores more of the parameter space. For this case, a variety of schemes exist that combine features of grid search and random search. Two popular examples are jittered sampling and Latin hypercube sampling[203]. For the former, samples are randomly placed within regularly spaced intervals, while the latter does so while maintaining an even distribution in the parameter space.

Grid and random search are often used for initial exploration of a parameter space to seed subsequent optimization with more advanced algorithms. An example for this is shown in Figure 15 (a), where grid search is combined with the downhill simplex method discussed in the next section.

4.3. Downhill simplex method and gradient-based algorithms

In the downhill simplex method, also known as the Nelder–Mead algorithm[204], an array of $(n+1)$ input parameter sets from an n-dimensional space is evaluated to get the corresponding function values. The worst-performing evaluation point is modified at each iteration by translating it towards or through the center of the simplex. This continues until the simplex converges to a point at which the function is optimized. The method is simple and efficient, which is why it is popular in various optimization settings. The convergence speed crucially depends on the initial choice of the simplex, with a large distance between input parameters leading to a more global optimization, while small simplex settings result in local optimization.

In the limit of small simplex size, the Nelder–Mead algorithm is conceptually related to gradient-based methods for optimization. The latter are based on the concept of using the gradient of the objective function to find the direction of the steepest slope. The objective function is then minimized along this direction using a suitable algorithm, such as gradient descent. Momentum descent is a popular variation of gradient descent where the gradient of the function is multiplied by a value, in analogy to physics called momentum, before being subtracted from the current position. This can help the algorithm converge to the local minimum faster. These methods typically require more and smaller iterations than the downhill simplex method, but can be more accurate.

In both cases, measurement noise can easily result in a wrong estimation of the gradient. In the setting of laser–plasma experiments, it is therefore important to reduce such noise, for example, by taking several measurements at the same position. While this may be possible when working with high-repetition-rate systems, as was demonstrated by Dann *et al.*[195], gradient-based methods are in general less suitable to be used in high-power-laser settings. Other popular derivative-based algorithms are the conjugate gradient (CG) method, the quasi-Newton (QN) method and the limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) method, all of which are discussed by Nocedal and Wright[151].

4.4. Genetic algorithms

One of the first families of algorithms to find application in field LPA was genetic algorithms. As a sub-class of evolutionary methods, these nature-inspired algorithms start with a pre-defined or random set, called the population, of measurements for different input settings. Each free parameter is

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

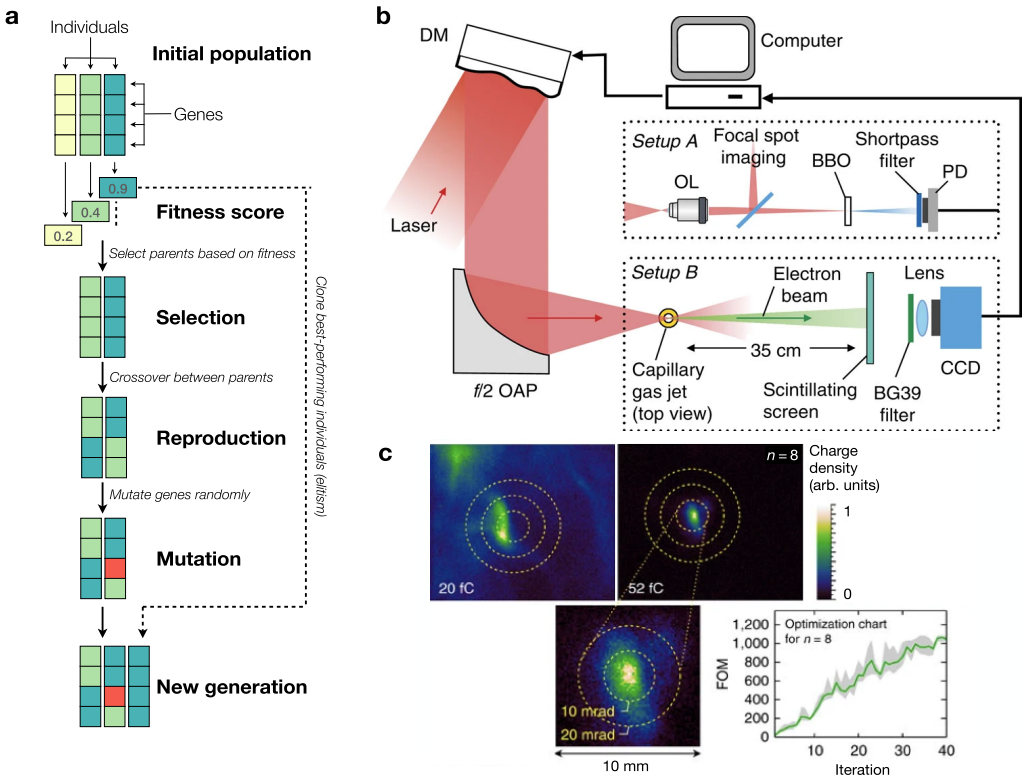

Figure 13. Genetic algorithm optimization. (a) Basic working principle of a genetic algorithm. (b) Sketch of a feedback-optimized LWFA via genetic algorithm. (c) Optimized electron beam spatial profiles using different figures of merit. Subfigures (b) and (c) adapted from Ref. [194].

called a gene and, similar to natural evolution, these genes can either crossover between most successful settings or randomly change (mutate). This process is guided by a so-called fitness function, which is designed to yield a single-valued figure of merit that is aligned with the optimization goal. Depending on the objective, the individual measurements are ranked from most to least fit. The fittest 'individuals' are then used to spawn a new generation of 'children', that is, a new set of measurements based on crossover and mutation of the parent genes. A popular variation in genetic algorithms is differential evolution[205], which employs a different type of crossover. Instead of crossing over two parents to create a child, differential evolution uses three parent measurements. The child is then created by adding a weighted difference between the parents to a random parent. This process is repeated until a new generation is created; see Figure 15(b) for an example.

One particular strength of genetic algorithms compared to many other optimization methods is their ability to perform multi-objective optimization, that is, when multiple, potentially conflicting, objectives are to be optimized. A popular example would be non-dominated sorting genetic algorithms (NSGAs)[206]. Here the name-giving sorting technique ranks the individuals by their population dominance, that is, how many other individuals in the population the individual dominates. Other multi-objective approaches are, for instance, based on optimizing niches, similar-valued regions of the Pareto front[207].

It should be noted that genetic algorithms intrinsically operate on population batches and not on a single individual. While this can be beneficial for parallel processing in simulations, it can make it more difficult to employ in an online optimization setup.

Genetic algorithms have been used since the early 2000s to control high-harmonic generation[208,209], cluster dynamics[210,211] and ion acceleration[212]. An influential example in the context of high-power lasers was published by He *et al.* in 2015[194], and a sketch of their feedback-looped experimental setup is presented in Figure 13. In the paper, a genetic algorithm is used to optimize various parameters of a laser–plasma accelerator, namely the electron beam angular profile, energy distribution, transverse emittance and optical pulse compression. This was done by controlling the voltage on 37 actuators of a deformable mirror (DM), which was used to shape a laser pulse before it entered a gas jet target. The genetic algorithm was initialized with a population of 100 individuals and each subsequent generation was generated based on the 10 fittest individuals[213]. A similar experiment was performed

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

using self-modulated LWFA[214] driven by a mid-infrared laser pulse. The electron beam charge, energy spectra and beam pointing have been optimized. The combination of genetic algorithm and DM has been applied to other high-power laser experiments as well[215–219].

Streeter *et al.*[220] used a similar technique to optimize ultra-fast heating of atomic clusters at joule-level energies, but instead of controlling the spatial wavefront, they controlled the spectral phase up to its fourth order. The genetic algorithm was based on a population of 15 individuals and 4−8 children generations were evaluated. This method can also be used for optimizing other laser parameters, such as focal spot size, focus position and chirp [195].

Another example is the use of differential evolution for optimization of the laser pulse duration in a SPIDER and DAZZLER feedback loop, as presented by Peceli and Mazurek[221]. The genetic algorithm method has also been employed in ion acceleration in a laser–plasma accelerator, where Smith *et al.*[222] optimized the conversion efficiency from laser energy to ion energy by exploring thousands of target density profiles in 1D particle-in-cell (PIC) simulations.

4.5. Bayesian optimization

BO[223,224] is a model-based global optimization method that uses probabilistic modeling, which was discussed in Section 2.1.3. The strength of BO lies in its efficiency with regard to the number of evaluations. This is particularly useful for problems with comparably high evaluation costs or long evaluation times. To achieve this, BO uses the probabilistic surrogate model to make predictions about the behavior of the system at new input parameter settings, providing both a value estimate and an uncertainty.

At the core of BO lies what is called the acquisition function, a pre-defined function that suggests the next points to probe on a probabilistic model. The latter is usually[9] a GP fitted from training points (see Section 2.1.4), thus providing a cheap-to-evaluate surrogate function. A simple and intuitive example of an acquisition function is the upper confidence bound:

$$\mathrm{UCB}(x) = \mu(x) + \kappa\sigma(x), \tag{27}$$

which weighs the mean prediction μ versus the variance prediction σ, with a user-chosen hyperparameter κ. For $\kappa = 0$ the optimization will act entirely exploitatively, that is, it will move to the position of the highest expected return, whereas a large κ incentivizes the reduction of the uncertainty and exploration of the parameter space. Other common acquisition functions are the expected improvement[225], knowledge gradient[226] and entropy search[227]. As the surrogate model can be probed at near-negligible evaluation cost, this optimization can be performed using numerical optimization methods, such as the gradient-based methods discussed in Section 4.3. The position of the acquisition function's optimum is then used as an input parameter setting to evaluate the actual system. This process is repeated until some convergence criteria are achieved, a pre-defined number of iterations is reached or the allocated resources have been otherwise exhausted.

BO provides a very flexible framework that can be further adapted to various different optimization settings. For instance, it has proven to be a valid alternative to evolutionary methods when it comes to solving multi-objective optimization problems. The importance of this method for LPA stems from the fact that many optimization goals, such as beam energy and beam charge, are conflicting in nature and require the definition of a trade-off. The goal of Pareto optimization is to find the Pareto front, which is a surface in the output objective space that comprises all the non-dominated solutions (see Section 4.1.2). A common metric that is used to measure the closeness of a set of points to the Pareto-optimal points is the hypervolume. The BO algorithm works by using the expected hypervolume improvement[228] to increase the extent of the current non-dominated solutions, thus optimizing all possible combinations of individual objectives. Note that the Pareto front, like the global optimum of single-objective optimization, is usually not known *a priori*.

Another possible way to extend BO is to incorporate different information sources[229]. This is often done by adding an additional information input dimension to the data model. This method is often referred to as multi-task (MT) or multi-fidelity (MF) BO. Both terms are used somewhat interchangeably in the literature, although MT often refers to optimization with entirely different systems (codes, etc.), whereas MF focuses on different fidelity (resolution, etc.) of the same information source. The core concept behind these methods is that the acquisition function not only encodes the objective, but also minimizes the measurement cost. These multi-information-source methods have the potential to speed up optimization significantly. They can also be combined with multi-objective optimization, as shown by Irshad *et al.*[199].

A first demonstration of BO in the context of LPA was presented by Lehe[230], who used the method to determine the injection threshold in a set of PIC simulations. The use of BO in experiments was pioneered by Shalloo *et al.*[196] (Figure 14), who demonstrated optimization

[9] While most work on BO is done using GP regression, the method is in principle model agnostic. This means that other types of (probabilistic) surrogate models of the system can be employed, such as decision trees (see Section 2.1.5) or deep neural networks (see Section 2.1.6).

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

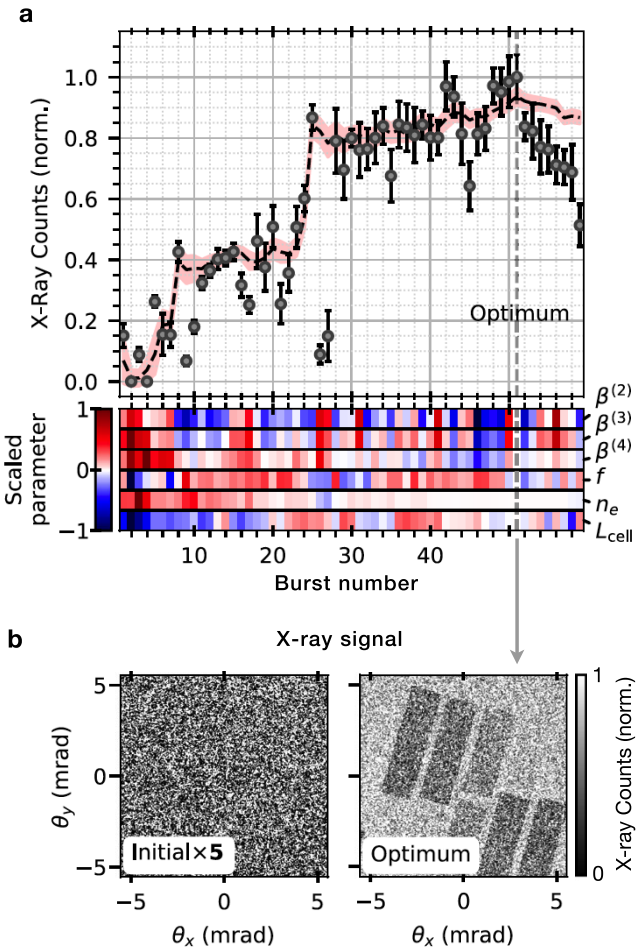

Figure 14. Bayesian optimization of a laser–plasma X-ray source. (a) The objective function (X-ray counts) as a function of iteration number (top) and the variation of the control parameters (bottom) during optimization. (b) X-ray images obtained for the initial (bottom) and optimal (top) settings. Adapted from Ref. [196].

of electron and X-ray beam properties from LWFA by automated control of laser and plasma control parameters. Another work by Jalas *et al.*[197] focused on improving the spectral charge density using the objective function $\sqrt{Q}\tilde{E}/E_{\mathrm{MAD}}$, thus incorporating the beam charge Q, the median energy $\tilde{E}$ and the energy spread defined here by the median absolute deviation E_{MAD}. In contrast to Shalloo *et al.*, they used shot-to-shot measurements of the control parameters to train the model rather than relying on the accuracy of their controllers, reducing the level of output variation attributed purely to stochasticity. BO has also been applied to the optimization of laser-driven ion acceleration in numerical simulations[231] and experiments[232]. A first implementation of multi-objective optimization in numerical simulations of plasma acceleration was published by Irshad *et al.*[198], who showed that multi-objective optimization can lead to superior performance compared with manually trying

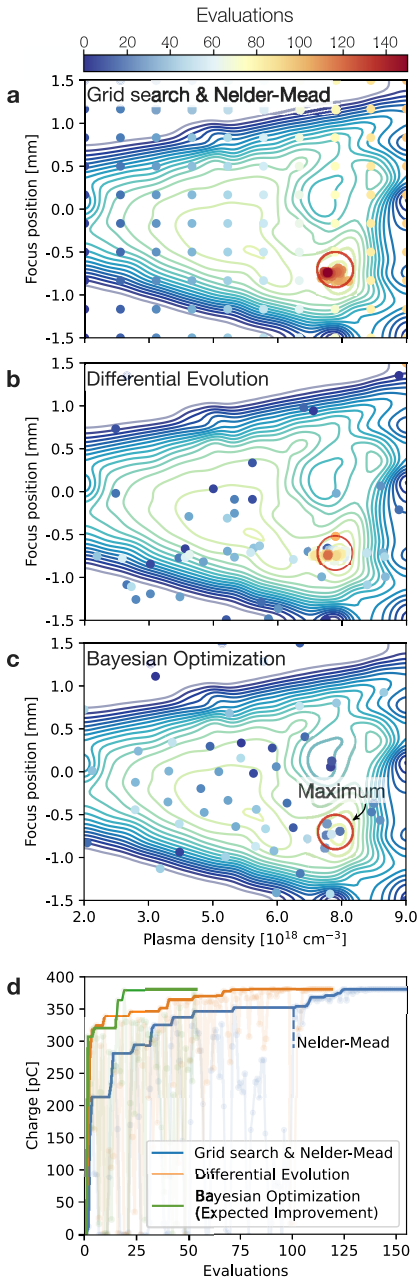

Figure 15. Illustration of different optimization strategies for a non-trivial 2D system, here based on a simulated laser wakefield accelerator with laser focus and plasma density as free parameters. The total beam charge, shown as contour lines in plots (a)–(c) serves as the optimization goal. The position of the optimum is marked by a red circle, located at a focus position of -0.74 mm and a plasma density of 8×10^{18} cm^{-3}. In panel (a), a grid search strategy with subsequent local optimization using the downhill simplex (Nelder–Mead) algorithm is shown. Panel (b) illustrates differential evolution and (c) is based on Bayesian optimization using the common expected improvement acquisition function. The performance for all three examples is compared in panel (d). It shows the typical behavior that Bayesian optimization needs the least and the grid search requires the most iterations. The local search via the Nelder–Mead algorithm converges within some 20 iterations, but requires a good initial guess (here provided by the grid search). Individual evaluations are shown as shaded dots. Note how the Bayesian optimization starts exploring once it has found the maximum, whereas the evolutionary algorithm tends more towards exploitation around the so-far best value. This behavior is extreme for the local Nelder–Mead optimizer, which only aims to exploit and maximize to local optimum.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

different trade-off definitions or settings for the individual objectives, in this case, the beam charge, energy spread and distance to a target energy. An example of MT BO in laser–plasma simulations was recently published by Pousa *et al.*[233], who combined the Wake-T and FBPIC codes, while Irshad *et al.*[198] used the FBPIC code at different resolutions for MF optimization.

4.6. Reinforcement learning

Reinforcement learning (RL)[234] differs fundamentally from the optimization methods discussed so far. RL is a method of learning the optimal behavior (the policy π) to control a system. The learning occurs via repeated trial and error, called episodes, where each episode is started in an initial state of the system, and then the agent (i.e., the optimizer) interacts with the system according to its policy. The agent then receives a reward signal, which is a scalar value that indicates the success of the current episode and its goal is to maximize the expected reward, analogous to the objective function in other optimization methods. The agent is said to learn when it is able to improve its policy to achieve a higher expected reward.

The policy itself has traditionally been represented using a Markov decision process (MDP), but in recent years deep reinforcement learning (DRL) has become widely used, in which the policy is represented using deep neural networks[235,236]. However, while we commonly update weights and biases via back-propagation in supervised deep learning, the learning in DRL is done in an unsupervised way. Indeed, while the agent is trying to learn the optimal policy to maximize the reward signal, the reward signal itself is unknown to the agent. The agent only knows the reward signal at the end of the episode, so it is not possible to perform back-propagation. Instead, the policy network can, for instance, be updated using evolutionary strategies (see Section 4.4), where the agents achieving the highest reward are selected to create a new generation of agents. Another common approach is to use a so-called actor–critic strategy[237], where a second network is introduced, called the critic. At the end of each episode, the critic is trained to estimate the expected long-term reward from the current state, called the value. This expected reward signal is then used to train the actor network to adjust the policy. The policy gradient[238] is a widely used algorithm for training the policy network using the critic network to calculate the expected reward signal.

RL algorithms can be further divided into two main classes: model-free and model-based learning. Model-free methods learn as discussed above directly by trial and error, only implicitly learning about the environment. Model-based methods, on the other hand, explicitly build a model of the environment, which can be used for both planning and learning (somewhat similar to optimization using surrogate models discussed earlier). The arguably most popular method to build models in RL is again the use of neural networks, as they can learn complex, nonlinear relationships and are also capable of learning from streaming data, which is essential in RL. A crucial advantage of the model-based approach is that it can drastically speed up training, although performance is always limited by the quality of the model. In the case of real-life systems this is sometimes referred to as the 'reality gap'.

One crucial advantage of RL is that once the training process is completed, the computational requirements of

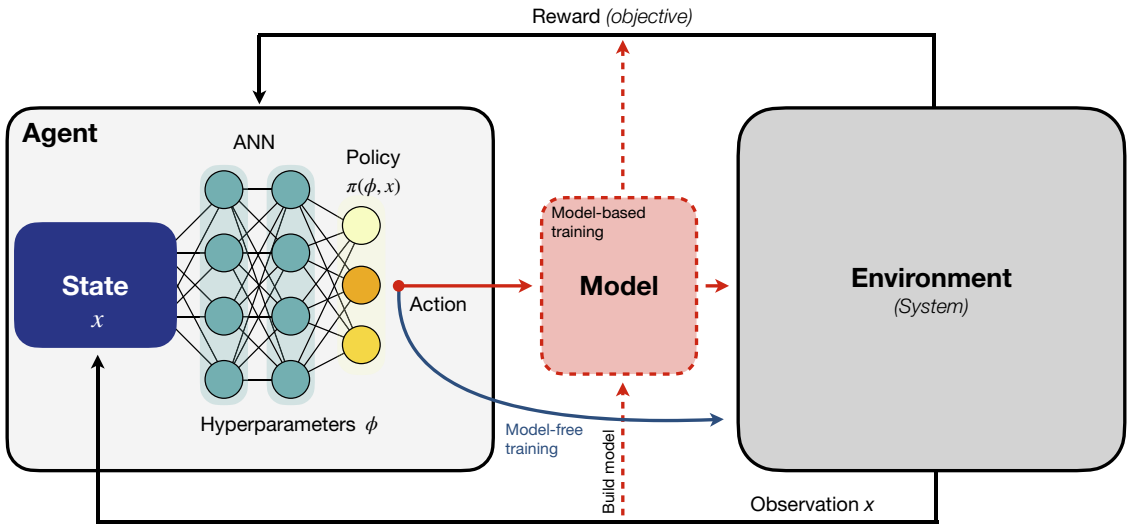

Figure 16. Sketch of deep reinforcement learning. The agent, which consists of a policy and a learning algorithm that updates the policy, sends an action to the environment. In the case of model-based reinforcement learning, the action is sent to the model, which is then applied to the environment. Upon the action to the environment, an observation is made and sent back to the agent as a reward. The reward is used to update the policy via the learning algorithm in the agent, which leads to an action in the next iteration.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

running an optimization are heavily reduced. A simplified representation of the RL workflow is sketched in Figure 16.

An example of an RL application is the work of Kain *et al.*[239] for trajectory optimization in the Advanced Wakefield (AWAKE) experiment in plasma wakefield acceleration, which found the optimum in just a few iterations based on 300 iterations of training. There are many other examples for the use of RL at accelerator facilities, for example, Refs. [240–243].

5. Unsupervised learning

In this section we are going to discuss unsupervised learning techniques for exploratory data analysis. The term 'unsupervised' refers to the case where one does not have access to training labels, and therefore the aim is not to find a mapping between training data and labels, as is often the case for deep learning. Rather, the aim is to detect relationships between features of the data.

For high-power laser experiments, many parameters will be coupled in some way such that there are correlations between different measurable input quantities. For example, increasing the laser energy in the amplifier chain of a high-power laser can affect the laser spectrum or beam profile. To understand the effect a change to any one of these parameters will have, it is important to consider their correlation. However, an experimental setup can easily involve tens of parameters and interpreting correlations becomes increasingly difficult. In this context, it can be useful to distill the most important (combinations of) parameters in a process called dimensionality reduction. The same method also plays a crucial role in efficient data compression, which is becoming increasingly important due to the large amount of data produced in both experiments and simulation. These methods are also closely related to pattern recognition, which addresses the issue of automatically discovering patterns in data.

5.1. Clustering

Data clustering is a machine learning task of finding similar data points and dividing them into groups, even if the data points are not labeled. This can, for instance, be useful to separate a signal from the background in physics experiments.

A popular centroid-based clustering algorithm is the K-means algorithm, which consists of two steps. First, the algorithm randomly assigns a cluster label to each point. Then, in the second step, it calculates the center point of each cluster and re-assigns the cluster label to each point based on the proximity to the cluster center. This process is repeated until the cluster assignment does not change anymore. The K in the algorithm's name represents the number of clusters,

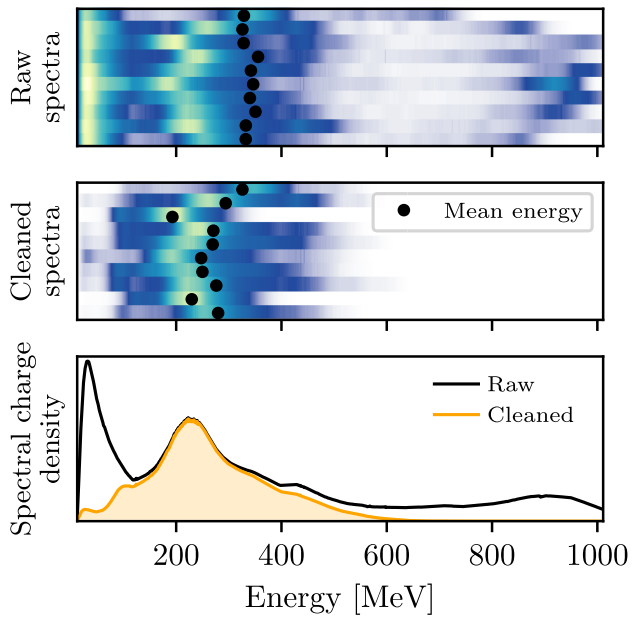

Figure 17. Data treatment using a Gaussian mixture model (GMM). *Top:* 10 consecutive shots from a laser wakefield accelerator. *Middle:* the same shots using a GMM to isolate the spectral peak at around 250 MeV. *Bottom:* average spectra with and without GMM cleaning. Adapted from Ref. [245].

which can be guessed or – more quantitatively – be estimated using methods such as the 'silhouette' value or the 'elbow method'[244]. As the method is quite sensitive to the initial random choice of the cluster assignment, it is often run several times with different initial choices to find the optimal classification.

In contrast to centroid-based clustering, in which each cluster is defined by a center point, in distribution-based clustering, each cluster is defined by a (multivariate) probability distribution. In the simplest case this can be a Gaussian distribution with a certain mean and variance for each cluster. More advanced methods use a Gaussian mixture model (GMM), in which each cluster is represented as a combination of Gaussian distributions. A popular distribution-based clustering method is the EM algorithm[155].

An example for the application of a GMM in data processing is shown in Figure 17. There a number of energy spectra from a laser wakefield accelerator are displayed. As the spectra exhibit multiple peaks, standard metrics such as the mean and standard deviation are not characteristic of either the peaks' positions or their widths. To avoid this problem a mixture model is used that isolates the spectral peaks. To this end, a combination of Gaussian distributions is fitted to the data and then a spectral bin is assigned with a certain probability to each distribution.

5.2. Correlation analysis

A simple method for exploring correlations is the correlation matrix, which is a type of matrix that is used to measure

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

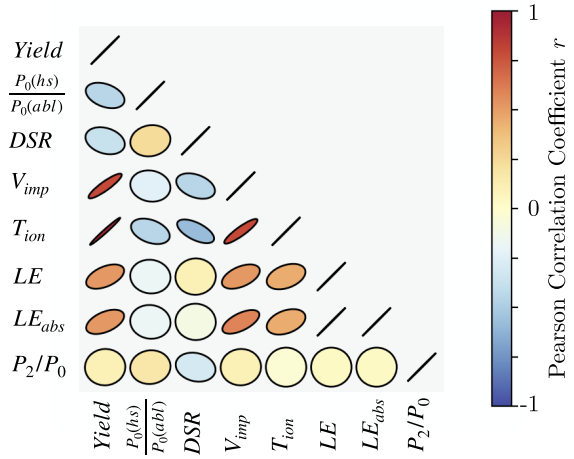

Figure 18. Correlogram – a visualization of the correlation matrix – of different variables versus yield at the NIF. Color indicates the value of the correlation coefficient. In this particular representation the correlation is also encoded in the shape and angle of the ellipses, helping intuitive understanding. The strongest correlation to the fusion yield is observed with the implosion velocity V_{imp} and the ion temperature T_{ion}. There is also a clear anti-correlation observable between the down-scattered ratio (DSR) and T_{ion} and, in accordance with the previously stated correlation of T_{ion} and yield, a weak anti-correlation of the DSR and yield. Note that all variables perfectly correlate with themselves by definition. Plot was generated based on data presented by Hsu *et al.*[96]. Further explanation (labels, etc.) can be found therein.

the relationship between two or more variables. We can calculate its coefficients, also known as Pearson correlation coefficients, on a set of n measurements of each pair of parameters x_i and y_i as follows:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}, \tag{28}$$

where $\bar{x}$ and $\bar{y}$ are the mean values of variables x and y, respectively. The correlation coefficient r is a number between –1 and 1. A value of 1 means that two variables are perfectly correlated, while a value of –1 means that two variables are perfectly anti-correlated. A value of 0 means that there is no correlation between two variables.

The correlation matrix and its visualization, sometimes called a correlogram, allow for a quick way to look for interesting and unexpected correlations. An example of this is shown in Figure 18. Note that by reducing correlations to a single linear term one can miss more subtle or complex relationships between variables. For such cases more general measures of correlation exist, such as the Spearman correlation coefficient that measures how well two variables can be described by any monotonic function.

5.3. Dimensionality reduction

Many datasets contain high-dimensional data but are governed by a few important underlying parameters. Signal decomposition and dimensionality reduction are processes that reduce the dimensionality of the data by separating a signal into its components or projecting it onto a lower-dimensional subspace so that the essential structure of the data is preserved. There are many ways to perform dimensionality reduction, two of the most common ones being principal component analysis (PCA) and AEs.

PCA is a very popular linear transformation technique that is used to convert a set of observations of possibly correlated variables into a (smaller) set of values of linearly uncorrelated variables, called the principal components. This transformation is defined in such a way that the first principal component has the largest possible variance, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. One method to perform PCA is to use SVD, which is used to decompose the matrix of data into a set of eigenvectors and eigenvalues. PCA shares a close relationship with correlation analysis, as the eigenvectors of the correlation matrix match those of the covariance matrix, which is utilized in defining PCA. In addition, the eigenvalues of the correlation matrix equate to the squared eigenvalues of the covariance matrix, provided that the data have been normalized. Kernel PCA[246] is an extension of PCA that uses a nonlinear transformation of the data to obtain the principal components. A relatively new variation, with some relation to the priorly discussed CS (Section 3.4), is robust principal component analysis (RPCA)[247]. RPCA is a modification of the original algorithm, which is better suitable to handle the presence of outliers in datasets. PCA should not be confused with the similarly named independent component analysis (ICA)[248], which is a popular technique to decompose a multivariate signal into a sum of statistically independent non-Gaussian signals.

There are also many neural network approaches to dimensionality reduction, one of the most popular ones being AEs. The purpose of an AE is to learn an approximation to the identity function, that is, a function that reproduces the input. In a standard AE, a neutral network is created

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

with an intermediate bottleneck layer with a reduced number of nodes, known as the latent space. During training, the neural network hyperparameters are optimized so that the output matches the input as closely as possible, typically by minimizing the MSE. Due to the bottleneck, the AE automatically discovers an efficient representation for the data in the latent space. The hidden layers up to the latent space are known as the decoder and the hidden layers from the latent space to the output layer are the encoder. The encoder can then be used separately to perform dimension reduction, equivalent to lossy data compression. With the corresponding decoder, an approximation of the original data can be extracted from its latent space.

There exist many different types of AE architecture, a particularly popular one being variational autoencoders (VAEs). VAEs replace deterministic encoder–decoder layers with a stochastic architecture (c.f. Figure 6) to allow the model to provide a probability distribution over the latent space. As a result, a VAE's latent space is smooth and continuous in contrast to a standard AE's latent space, which is discrete. This allows VAEs to also generate new data by sampling from the latent space.

AEs have also shown a strong potential as advanced compression techniques that can be highly adapted to many kinds of inputs. In this case, one trains an AE model to find an approximation of the identity function for some raw data. After training, the raw data are sent through the encoding layer; only the dimensionality reduces and highly compressed latent space representation in the bottleneck layer is saved. Decompression is achieved by sending the data through the decoding layer. This method is not only relevant to reduce disc space occupied by data; AEs are nowadays frequently used as an integral part of complex machine learning tasks, where the latent space is used as a lower-dimensional input for, for example, a diffusion network (as part of what is called a latent diffusion model[249]) or a Bayesian optimizer[250].

An example of the modeling using the latent space of an AE was recently published by Willmann *et al.*[251]. Working on the problem of simulating shadowgrams from plasma probing, they used an AE to reduce the 3D input data and then trained a small four-layer perceptron network to learn how to approximate the shadowgram formation. An example of pure compression was recently presented by Stiller *et al.*[252], who applied an AE to compress data from PIC simulations, showing promising first results.

6. Image analysis

In the previous section we discussed how to analyze datasets by looking for correlations or compressed representations. A closely related group of tasks occurs when dealing with image data, that is, image recognition or classification, object detection and segmentation. While the methods in the previous section dealt with features learnt from the data itself, the techniques discussed in this section aim to identify or locate specific features in our data (in particular images). As such, these are all considered supervised learning methods.

6.1. Classification

The classification problem in machine learning is the problem of correctly labeling a data point from a given set of data points with the correct label. The data points are labeled with a categorical class label, such as 'cat', 'dog', 'electron' or 'ion' (see Figure 19(a)).

In the following we are going to briefly discuss some of the most important machine learning techniques used for classification. It should be noted that classification is closely related to regression, with the main difference being essentially that the model's output is a class value instead of a value prediction. As such, methods for working in regression can in general also be applied to classification tasks. One example is the decision tree method, which we already discussed in Section 2.1.5.

6.1.1. Support vector machines

Support vector machines (SVMs) are a popular set of machine learning methods used primarily in classification and recognition. For a simple binary classification task, a SVM draws a hyperplane that divides all data points of one category from all data points of the other category. While there could be many hyperplanes, an SVM looks for the optimal hyperplane that best separates the dataset by maximizing the margin between the hyperplane and the data points in both categories. The points that locate right

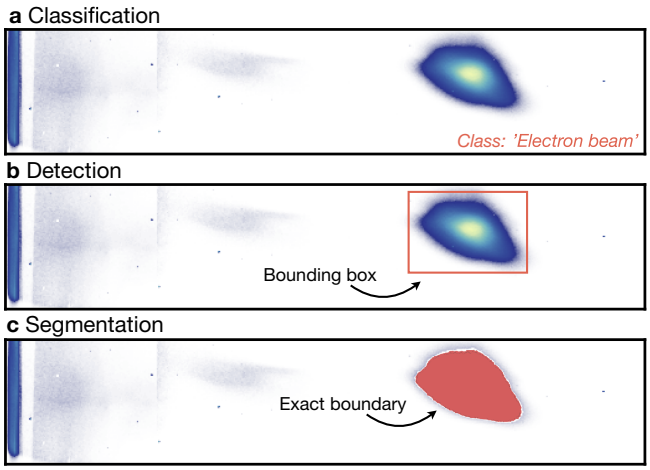

Figure 19. Illustration of common computer vision tasks. (a) Classification is used to assign (multiple) labels to data. (b) Detection goes a step further and adds bounding boxes. (c) Segmentation provides pixel maps with exact boundaries of the object or feature.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

on the margin are called 'supporting vectors'. For a dataset with more than two categories, classification is performed by dividing the multi-classification problem into several binary classification problems and then finding a hyperplane for each. In practice, the data points in two categories can mix together so that they cannot be clearly divided by a linear hyperplane. For such nonlinear classification tasks, the kernel trick is used to compute the nonlinear features of the data points and map them to a high-dimensional space, so that a hyperplane can be found to separate the high-dimensional space. The hyperplane will then be mapped back to the original space.

The ideal application scenario for an SVM is for datasets with small samples but high dimensions. The choice of various kernel functions also adds to the flexibility of this method. However, an SVM would be very computationally expensive for large datasets. Besides, its accuracy can significantly decrease when analyzing datasets with large noise levels, as the hyperplanes cannot be defined precisely. Therefore, especially in the context of high-power laser experiments, one has to be cautious when applying SVMs if there are considerable stability issues.

6.1.2. Convolutional neural networks

CNNs are a type of neural network (cf. Section 2.1.6) that is particularly well-suited for image classification[253], but are also used in various other problems.

Such a network is composed of sequential convolutional layers, in each of which an $N \times N$ kernel (or 'filter' matrix) is convolved with the output of the previous layer. This operation is done by sliding the kernel over the input image, and each pixel in the output layer is calculated by the dot product of the kernel with a sub-section of the input image centered around the corresponding pixel. Resultingly, convolutional layers are capable of detecting local patterns in the $N \times N$ region of the kernel. The kernel is parameterized with weights, which are learned via back-propagation as in a regular neural network. Within a layer, there can also be multiple channels of the output, practically thought of as multiple kernels being passed over the image, allowing for different features to be detected. The early layers of a CNN detect simple structures such as edges, but by adding multiple layers with varying kernel sizes, the network can perform high-level abstraction in order to detect complicated patterns.

The convolutional layer makes the CNN very efficient for image classification. It allows the network to learn translation-invariant features – a feature learned at a certain position of an image can be recognized at any position on the same image.

In order to detect patterns that are non-local in the image, pooling is often applied. There exist many schemes of pooling, but the general concept is to take a set of pixels in the input and to apply some operation that turns them into 1 pixel. Examples include max-pooling (taking the maximum of the set of pixels) and average pooling. This operation decreases the dimensions of the image, and therefore allows a subsequent convolutional layer with the same $N \times N$ kernel to detect features that were much further apart in the original image. Typical CNNs will use multiple pooling layers to decrease the dimensions until a kernel can nearly span the whole image to detect any non-local patterns. The output is then flattened and several fully connected layers can be used to manipulate the data for the relevant (i.e., regression or classification) task.

While the use of deeper CNNs with an increasing number of layers tends to improve performance[254], architectures can suffer from unstable gradients in training via back-propagation[255], showing that some deeper architectures are not as easy to optimize. One particularly influential solution to this problem was the introduction of the residual shortcut, where the input to a block is added to the output of the block. In back-propagation, this enables the 'skipping' of layers, effectively simplifying the network and leading to better convergence. This was first proposed and implemented in ResNet [256], which has since become a standard for deep CNN architectures[257], with any number of variations. It should be mentioned though that a number of competitive networks without residual shortcuts exist, for example, AlexNet [258].

6.2. Object detection

In the context of image data, object detection can be seen as an extension of classification, yielding both a label as in classification tasks and the position of that object, illustrated in Figure 19(b). The main challenge is that complex images can contain many objects with different features, while the number of objects can also differ from one image to another. Therefore, object detection techniques require a certain flexibility regarding their structure.

The Viola–Jones algorithm[259] is one of the most popular object detection algorithms from the 2000s, pre-dating the recent network-based object detectors. Generally, it involves detecting objects by looking at the image as a set of small patches, and identifying so-called Haar-like features. The latter are patterns that occur frequently in images, and can be used to distinguish between different objects. The Viola–Jones algorithm detects objects by first analyzing an image at different scales. For each scale, it looks for features by scanning the image with a sliding window, and for each window, it computes a list of features used to identify the object. If the object is detected, the algorithm returns the bounding box of the object. The Viola–Jones framework does not allow for simultaneous classification and instead requires a subsequent classifier, such as an SVM, for that task. Compared to its network-based alternatives, the Viola–Jones algorithm is worse in terms of precision but better in

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

terms of computational cost, allowing real-time detection at high frame rates[260].

Object detection networks are more complicated than a regular CNN, because the length of the output layer of the neural network cannot be pre-defined due to the unknown number of objects on an image. A possible solution is to divide the image into many regions and to construct a CNN for each region, but that leads to significant computational cost. Two families of networks are developed to detect objects at reasonable computational cost. The region proposal network (RPN) takes an image as input and outputs a set of proposals for possible objects in the image. A CNN is used to find features in each possible region and to classify the feature into known category. The RPN is trained to maximize the overlap between the proposals and the ground truth objects. The state-of-the-art algorithm in this family is the Faster-RCNN[261].

An alternative to region-based CNNs is the YOLO[262] family. YOLO stands for 'You Only Look Once' as it only looks at the image once with a single neural network to make predictions. This is different from other object detection algorithms, which often employ many neural networks for the image. As a result, YOLO is typically much faster, allowing for real-time object detection. The disadvantage is that it is not as accurate as some of the other object detection algorithms. Nevertheless, the fast inference speed of YOLO is particularly appealing to high-power laser facilities with high-repetition-rate capability.

6.3. Segmentation

Semantic segmentation[263] is a related task in computer vision that seeks to create a pixel-by-pixel mapping of the input image to a class label, not only a bounding box as in object detection (see Figure 19(c)). By doing so, segmentation defines the exact boundary of the objects.

Many semantic segmentation architectures are based on FCNs[264] and have evolved considerably in recent years. Since FCNs suffered from the issue of semantic gaps, where the output has a much lower resolution than the input, skip connections were introduced to allow the gradient to back-propagate through the layers to improve the performance. Examples of such advanced network architectures are the U-Net[180] and the DeepLab network[263], which are based on ResNet-101. Both of these architectures use residual skip connections to maintain gradient flow. The advantage of semantic segmentation compared to standard object detection is that the network can easily localize multiple objects of the same class in an image. The disadvantage is that one needs to train a separate network for each class.

Related is instance segmentation[265], which goes a step further and distinguishes each individual instance of an object, not just the class. Instance segmentation is a significant challenge, as it requires the network to be able to distinguish between two instances of the same object, for example, two cats. Instance segmentation architectures are typically based on mask R-CNN[266], which combines a CNN with a region-based convolutional neural network (R-CNN)[267] and an FCN[264]. Note that the mask R-CNN can be used for both semantic and instance segmentation.

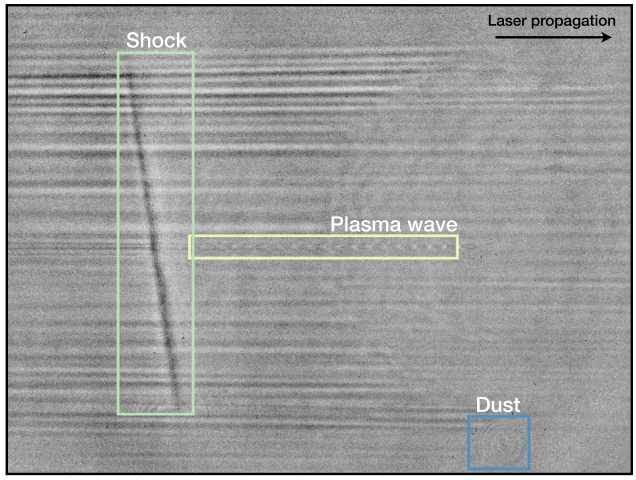

Figure 20. Application of object detection to a few-cycle shadowgram of a plasma wave: the plasma wave, the shadowgram of a hydrodynamic shock and the diffraction pattern caused by dust are correctly identified by the object detector and located with bounding boxes. Adapted from Ref. [273].

One of the prime examples for machine-learning-assisted image analysis is the automated detection and classification of laser damage. Researchers at the NIF have pioneered this approach with several works on neural networks for damage classification. For instance, Amorin *et al.*[268] trained CNNs based on the AlexNet and Inception architectures to distinguish between different kinds of laser damage. Another example for the use of both SVM- and CNN-based classification in high-power laser systems was recently presented by Pascu[269], who used both techniques for (supervised) anomaly detection in a laser beam profile at the ELI-NP facility. Chu *et al.*[270] presented a first application of image segmentation to locate laser-induced defects on optics in real time using a U-Net. Ben Soltane *et al.*[271] recently presented a deep learning pipeline to estimate the size of damages in glass windows at the Laser Mégajoule (LMJ) facility, using a similar U-Net architecture for segmentation. Li *et al.*[272] combined damage detection via a deep neural network with postprocessing to position laser damages in 3D space. The axial distance between the damage site and the imaging system is obtained numerically by the principle of holographic focusing. More examples for applications of object detection in a high-power laser laboratory have been reported in the work of Lin *et al.*[273]. In addition to the aforementioned case of optical damages in a high-

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

power laser beamline, the authors fine-tuned the YOLO network for object detection in the few-cycle shadowgraphy of plasma waves and electron beam detection in an electron spectrometer. An example of the detected features in a shadowgram is presented in Figure 20. The position and size of the detected objects are used to determine information about the physical quantities, such as the plasma wavelength and plasma density distribution.

7. Discussion and conclusions

In this paper, we have presented an overview of techniques and recent developments in machine learning for laser–plasma physics. As we have seen, early proof-of-concept papers appeared in the late 1990s and early 2000s, but the computing power available at the time was typically not sufficient to make the approaches competitive with established methods or to reach a suitable level of accuracy. In the mid-2010s, a resurgence of interest in the field began, with an ever-increasing number of publications. A significant fraction of the papers that have been reviewed here are experimental in nature, especially regarding optimization (see Table 2). On one hand, this can be attributed to the increasing digitization of the laboratory environment, with control systems, data acquisition and other developments providing access to large amounts of data. On the other hand, the complexity of modern experiments acts as a catalyst for the development of automated data analysis and optimization techniques. Deployment of machine learning techniques in a real-world environment can however be challenging, for example, due to noise, jitter and drifts. This is certainly one of the reasons why the most advanced machine learning techniques, such as multi-objective optimization or deep CS, tend to be first tested with simulations.

Among the methods being employed we also observed some general trends. For instance, while genetic algorithms have been very popular in the past for global optimization, there has been an increasing number of papers focusing on BO. This is likely due to the fact that both simulations and experiments in the context of laser–plasma physics are very costly, making the use of Bayesian approaches more appealing. In most experimental settings, local optimization algorithms, such as gradient descent or the Nelder–Mead algorithm, are less suitable because of the large number of iterations needed and their sensitivity to noise. RL, especially in its model-free incarnation, suffers from the same issue, which explains why only very few examples of its use exist in adjacent research fields. While rather popular among data scientists due to their simplicity and interpretability, decision tree methods have not seen wide application in laser–plasma physics. In part, this is likely due to the fact that these methods are often considered to have more limited capabilities in comparison to neural networks, making it more attractive to directly use deep neural networks. In the context of ill-posed inverse problems it is to be expected that end-to-end neural networks or hybrid approaches will gradually replace traditional methods, such as regularization via total variation. That said, the simplicity and bias-free nature of least-squares methods are likely to ensure their continued popularity, at least in the context of easier to solve well-posed problems.

Much of the success of machine learning techniques stems from the fact that they are able to leverage prior knowledge, be it in the form of physical laws (e.g., via PINNs) or in the form of training data (e.g., via deep learning). Regarding the latter, the importance of preparing input data cannot be overstated. A popular saying in supervised learning is ‘garbage in, garbage out’, meaning that the quality of a model’s output heavily depends on the quality of the training data. Important steps are, for instance, pre-processing[276] (noise removal, normalization, etc.) and data augmentation[277] (rotation, shifts, etc.). The latter is of particular importance when dealing with experimental data, for which data acquisition is usually costly, making it challenging to acquire enough data to train a well-performing model. Furthermore, even when using regularization techniques, diversity of training data is very important to ensure good generalization capabilities and to avoid bias.

Two outstanding issues for the wide adoption of machine learning models in the laser–plasma community are interpretability and trustworthiness, both regarding the model itself and on the user side. While some machine learning models such as decision trees can be interpreted comparably easily, the inner workings of advanced models such as deep neural networks are often difficult to understand. This issue is amplified by the fact that integrated machine learning environments allow users to quickly build complex models without a thorough understanding of the underlying principles. We hope that this review will help to alleviate this issue, by providing a better understanding of the origin, capabilities and limitations of different machine learning techniques. Regarding trustworthiness, quantification of aleatoric uncertainty in training data and epistemic uncertainty of the model remains an important research area[278]. For instance, well-tested models may break down when exposed to unexpected input data, for example, due to drifts in experimental conditions or changes in the experimental setup. Such issues can for instance be addressed by incorporating uncertainty quantification into models to highlight unreliable predictions.

As our discussion has shown, there is an ever-increasing interest in data-driven and machine learning techniques within the community and we hope that our paper provides useful guidance for those starting to work in this rapidly evolving field. To facilitate some hands-on experimentation, we conclude with a short guide on how to get started in implementing the techniques we have discussed in this paper.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

Table 2. Summary of papers used as application examples in this review, sorted by year for each section.

Author, year	Problem type	ML technique	Sim.	Exp.	Research field
Humbird *et al.*, 2018[94]	Forward model	Neural net & decision tree	✓	✗	Inertial confinement fusion
Humbird *et al.*, 2018[95]	Forward model	Transfer learning	✓	✓	Inertial confinement fusion
Gonoskov *et al.*, 2019[106]	Forward model	Neural network	✓	✓	High-harmonic generation
Maier *et al.*, 2020[26]	Forward model	Linear regression	✓	✗	Laser wakefield acceleration
Kluth *et al.*, 2020[97]	Forward model	Autoencoder & DJINN	✓	✓	Inertial confinement fusion
Kirchen *et al.*, 2021[29]	Forward model	Neural network	✗	✓	Laser wakefield acceleration
Rodimkov *et al.*, 2021[107]	Forward model	Neural network	✓	✗	Noise robustness in PIC codes
Djordjević *et al.*, 2021[108]	Forward model	Neural network	✓	✗	Laser-ion acceleration
Watt, 2021[109]	Forward model	Neural network	✓	✗	Strong-field QED
McClarren *et al.*, 2021[110]	Forward model	Neural network	✓	✗	Inertial confinement fusion
Simpson *et al.*, 2021[111]	Forward model	Neural network	✗	✓	Laser–solid interaction
Streeter *et al.*, 2023[112]	Forward model	Neural network	✗	✓	Laser wakefield acceleration
Krumbügel *et al.*, 1996[186]	Inverse problem	Neural network	✗	✓	Spectral phase retrieval
Sidky *et al.*, 2005[274]	Inverse problem	EM algorithm	✗	✓	X-ray spectrum reconstruction
Döpp *et al.*, 2018[162]	Inverse problem	Statistical iterative reconstruction	✗	✓	X-ray tomography with betatron radiation
Huang *et al.*, 2014[171]	Inverse problem	Compressed sensing	✓	✗	ICF radiation analysis
Zahavy *et al.*, 2018[187]	Inverse problem	Neural network	✗	✓	Spectral phase retrieval
Hu *et al.*, 2020[188]	Inverse problem	Neural network	✗	✓	Wavefront measurement
Ma et al., 2020[173]	Inverse problem	Compressed sensing	✗	✓	Compton X-ray tomography
Li *et al.*, 2021[275]	Inverse problem	Compressed sensing	✓	✗	ICF radiation analysis
Howard *et al.*, 2023[192]	Inverse problem	Compressed sensing/deep unrolling	✓	✗	Hyperspectral phase imaging
Bartels *et al.*, 2000[208]	Optimization	Genetic algorithm	✗	✓	High-harmonic generation
Yoshitomi *et al.*, 2004[209]	Optimization	Genetic algorithm	✗	✓	High-harmonic generation
Zamith *et al.*, 2004[211]	Optimization	Genetic algorithm	✗	✓	Cluster dynamics
Yoshitomi *et al.*, 2004[209]	Optimization	Genetic algorithm	✗	✓	Cluster dynamics
Nayuki *et al.*, 2005[212]	Optimization	Genetic algorithm	✗	✓	Ion acceleration
He *et al.*, 2015[194,213]	Optimization	Genetic algorithm	✗	✓	Laser wakefield acceleration
Streeter *et al.*, 2018[220]	Optimization	Genetic algorithm	✗	✓	Cluster dynamics
Lin *et al.*, 2019[214]	Optimization	Genetic algorithm	✗	✓	Laser wakefield acceleration
Dann *et al.*, 2019[195]	Optimization	Genetic & Nelder–Mead algorithms	✗	✓	Laser wakefield acceleration
Shalloo *et al.*, 2020[196]	Optimization	Bayesian optimization	✗	✓	LWFA, betatron radiation
Smith *et al.*, 2020[222]	Optimization	Genetic algorithm	✓	✗	Laser-ion acceleration
Kain *et al.*, 2020[239]	Optimization	Reinforcement learning	✓	✓	Plasma wakefield acceleration
Jalas *et al.*, 2021[197]	Optimization	Bayesian optimization	✓	✓	Laser wakefield acceleration
Pousa *et al.*, 2022[233]	Optimization	Bayesian optimization	✓	✗	Laser wakefield acceleration
Dolier *et al.*, 2022[231]	Optimization	Bayesian optimization	✓	✗	Laser-ion acceleration
Irshad *et al.*, 2023[198]	Optimization	Bayesian optimization	✓	✗	Laser wakefield acceleration
Loughran *et al.*, 2023[232]	Optimization	Bayesian optimization	✗	✓	Laser-ion acceleration
Irshad *et al.*, 2023[245]	Optimization	Bayesian optimization	✗	✓	Laser wakefield acceleration
Chu *et al.*, 2019[270]	Image analysis	Neural network	✗	✓	Laser damage segmentation
Amorin *et al.*, 2019[268]	Image analysis	Neural network	✗	✓	Laser damage analysis
Li *et al.*, 2020[272]	Image analysis	Neural network	✗	✓	Laser damage detection in three dimensions
Hsu *et al.*, 2020[96]	Feature analysis	Six supervised learning methods	✗	✓	Inertial confinement fusion
Lin *et al.*, 2021[98]	Feature analysis	Four supervised learning methods	✗	✓	Laser wakefield acceleration
Willmann *et al.*, 2021[251]	Dimensionality reduction	Autoencoder	✓	✗	Laser wakefield acceleration
Stiller *et al.*, 2022[252]	Data compression	Autoencoder	✓	✗	Laser wakefield acceleration
Pascu, 2022[269]	Image analysis	SVM/neural network	✗	✓	Laser anomaly detection
Ben Soltane *et al.*, 2022[271]	Image analysis	Neural network	✗	✓	Laser damage segmentation
Lin *et al.*, 2023[273]	Image analysis	Neural network	✗	✓	Laser wakefield acceleration and damage detection

Most of these are readily implemented in several extensive libraries, Scikit-learn[279], TensorFlow[280] and PyTorch[281] being among the most popular ones. Each of these libraries has its own strengths and weaknesses. In particular, deep learning libraries such as TensorFlow and PyTorch are tailored for fast computations on graphics processing units (GPUs), whereas libraries such as Scikit-learn are designed for more general machine learning tasks. Higher level frameworks exist to facilitate the training of neural networks, for example, MLflow or PyTorch lightning.

The Darts library[282] contains implementations of various time series forecasting models, and also acts as a wrapper for numerous other libraries related to forecasting. Many numerical optimization algorithms, such as derivative-based meth-

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

ods and differential evolution, can be easily explored using the optimization library of SciPy[283]. While this includes, for instance, differential evolution, more sophisticated evolutionary algorithms such as multi-objective evolutionary methods require dedicated libraries such as pymoo[284] or PyGMO[285]. BO can for instance be implemented within Scikit-learn or using the experimentation platform Ax. The highly optimized BoTorch library[286] can be used for more advanced applications, including for instance multi-objective MF optimization. Some libraries are specifically tailored to hyperparameter optimization, such as the popular optuna library[287].

While all of the above examples are focused on Python as an underlying programming language, machine learning tasks can also be performed using many other programming platforms or languages, such as MATLAB or Julia. Jupyter notebooks provide a good starting point, and some online implementations, such as Google Colab, even give limited access to GPUs for training. The reader is encouraged to explore the various frameworks and libraries to find the one that best suits their needs.

Acknowledgements

The authors thank all participants of the *LPA Online Workshop on Control Systems and Machine Learning* for their contributions to the workshop, many of which are referenced throughout this manuscript. We particularly thank N. Hoffmann, S. Jalas, M. Kirchen, R. Shalloo and A. G. R. Thomas for helpful feedback and comments on this manuscript. The authors acknowledge the use of GPT-3[288] (text-davinci-003) in the copy-editing process of this manuscript.

References

1. C. Danson, D. Hillier, N. Hopps, and D. Neely, High Power Laser Sci. Eng. **3**, e3 (2015).
2. C. N. Danson, C. Haefner, J. Bromage, T. Butcher, J.-C. F. Chanteloup, E. A. Chowdhury, A. Galvanauskas, L. A. Gizzi, J. Hein, D. I. Hillier, N. W. Hopps, Y. Kato, E. A. Khazanov, R. Kodama, G. Korn, R. Li, Y. Li, J. Limpert, J. Ma, C. H. Nam, D. Neely, D. Papadopoulos, R. R. Penman, L. Qian, J. J. Rocca, A. A. Shaykin, C. W. Siders, C. Spindloe, S. Szatmári, R. M. G. M. Trines, J. Zhu, P. Zhu, and J. D. Zuegel, High Power Laser Sci. Eng. **7**, e54 (2019).
3. A. Pukhov, Rep. Prog. Phys. **66**, 47 (2003).
4. M. Marklund and P. K. Shukla, Rev. Mod. Phys. **78**, 591 (2006).
5. J. M. Cole, K. T. Behm, E. Gerstmayr, T. G. Blackburn, J. C. Wood, C. D. Baird, M. J. Duff, C. Harvey, A. Ilderton, A. S. Joglekar, K. Krushelnick, S. Kuschel, M. Marklund, P. McKenna, C. D. Murphy, K. Poder, C. P. Ridgers, G. M. Samarin, G. Sarri, D. R. Symes, A. G. R. Thomas, J. Warwick, M. Zepf, Z. Najmudin, and S. P. D. Mangles, Phys. Rev. X **8**, 011020 (2018).
6. K. Poder, M. Tamburini, G. Sarri, A. Di Piazza, S. Kuschel, C. D. Baird, K. Behm, S. Bohlen, J. M. Cole, D. J. Corvan, M. Duff, E. Gerstmayr, C. H. Keitel, K. Krushelnick, S. P. D. Mangles, P. McKenna, C. D. Murphy, Z. Najmudin, C. P. Ridgers, G. M. Samarin, D. R. Symes, A. G. R. Thomas, J. Warwick, and M. Zepf, Phys. Rev. X **8**, 031004 (2018).
7. T. G. Blackburn, Rev. Mod. Plasma Phys. **4**, 5 (2020).
8. F. Quéré and H. Vincenti, High Power Laser Sci. Eng. **9**, e6 (2021).
9. R. P. Drake, Phys. Plasmas **16**, 055501 (2009).
10. B. Mahieu, N. Jourdain, K. T. Phuoc, F. Dorchies, J.-P. Goddet, A. Lifschitz, P. Renaudin, and L. Lecherbourg, Nat. Commun. **9**, 3276 (2018).
11. B. Kettle, E. Gerstmayr, M. J. V. Streeter, F. Albert, R. A. Baggott, N. Bourgeois, J. M. Cole, S. Dann, K. Falk, I. G. González, A. E. Hussein, N. Lemos, N. C. Lopes, O. Lundh, Y. Ma, S. J. Rose, C. Spindloe, D. R. Symes, M. Šmíd, A. G. R. Thomas, R. Watt, and S. P. D. Mangles, Phys. Rev. Lett. **123**, 254801 (2019).
12. N. F. Beier, H. Allison, P. Efthimion, K. A. Flippo, L. Gao, S. B. Hansen, K. Hill, R. Hollinger, M. Logantha, Y. Musthafa, R. Nedbailo, V. Senthilkumaran, R. Shepherd, V. N. Shlyaptsev, H. Song, S. Wang, F. Dollar, J. J. Rocca, and A. E. Hussein, Phys. Rev. Lett. **129**, 135001 (2022).
13. E. Esarey, C. B. Schroeder, and W. P. Leemans, Rev. Mod. Phys. **81**, 1229 (2009.
14. V. Malka, Phys. Plasmas **19**, 055501 (2012).
15. S. M. Hooker, Nat. Photonics **7**, 775 (2013).
16. W. P. Leemans, B. Nagler, A. J. Gonsalves, C. Tóth, K. Nakamura, C. G. R. Geddes, E. Esarey, C. B. Schroeder, and S. M. Hooker, Nat. Phys. **2**, 696 (2006).
17. S. Kneip, S. R. Nagel, S. F. Martins, S. P. D. Mangles, C. Bellei, O. Chekhlov, R. J. Clarke, N. Delerue, E. J. Divall, G. Doucas, K. Ertel, F. Fiuza, R. Fonseca, P. Foster, S. J. Hawkes, C. J. Hooker, K. Krushelnick, W. B. Mori, C. A. J. Palmer, K. T. Phuoc, P. P. Rajeev, J. Schreiber, M. J. V. Streeter, D. Urner, J. Vieira, L. O. Silva, and Z. Najmudin, Phys. Rev. Lett. **103**, 035002 (2009).
18. C. E. Clayton, J. E. Ralph, F Albert, R. A. Fonseca, S. H. Glenzer, C. Joshi, W. Lu, K. A. Marsh, S. F. Martins, W. B. Mori, A. Pak, F. S. Tsung, B. B. Pollock, J. S. Ross, L. O. Silva, and D. H. Froula, Phys. Rev. Lett. **105**, 105003 (2010).
19. X. Wang, R. Zgadzaj, N. Fazel, Z. Li, S. A. Yi, X. Zhang, W. Henderson, Y.-Y. Chang, R. Korzekwa, H.-E. Tsai, C.-H. Pai, H. Quevedo, G. Dyer, E. Gaul, M. Martinez, A. C. Bernstein, T. Borger, M. Spinks, M. Donovan, V. Khudik, G. Shvets, T. Ditmire, and M. C. Downer, Nat. Commun. **4**, 1988 (2013).
20. W. P. Leemans, A. J. Gonsalves, H.-S. Mao, K. Nakamura, C. Benedetti, C. B. Schroeder, C. Tóth, J. Daniels, D. E. Mittelberger, S. S. Bulanov, J.-L. Vay, C. G. R. Geddes, and E. Esarey, Phys. Rev. Lett. **113**, 245002 (2014).
21. A. J. Gonsalves, K. Nakamura, J. Daniels, C. Benedetti, C. Pieronek, T. C. H. de Raadt, S. Steinke, J. H. Bin, S. S. Bulanov, J. van Tilborg, C. G. R. Geddes, C. B. Schroeder, C. Tóth, E. Esarey, K. Swanson, L. Fan-Chiang, G. Bagdasarov, N. Bobrova, V. Gasilov, G. Korn, P. Sasorov, and W. P. Leemans, Phys. Rev. Lett. **122**, 084801 (2019).
22. Z. H. He, B. Hou, J. A. Nees, J. H. Easter, J. Faure, K. Krushelnick, and A. G. R. Thomas, New J. Phys. **15**, 053016 (2013).
23. J. Faure, D. Gustas, D. Guénot, A. Vernier, F. Böhle, M. Ouillé, S. Haessler, R. Lopez-Martens, and A. Lifschitz, Plasma Phys. Controll. Fusion **61**, 014012 (2018).
24. L. Rovige, J. Huijts, I. Andriyash, A. Vernier, V. Tomkus, V. Girdauskas, G. Raciukaitis, J. Dudutis, V. Stankevic, P. Gecys, M. Ouille, Z. Cheng, R. Lopez-Martens, and J. Faure, Phys. Rev. Accel. Beams **23**, 093401 (2020).
25. F. Salehi, M. Le, L. Railing, M. Kolesik, and H. M. Milchberg, Phys. Rev. X **11**, 021055 (2021).
26. A. R. Maier, N. M. Delbos, T. Eichner, L. Hübner, S. Jalas, L. Jeppe, S. W. Jolly, M. Kirchen, V. Leroux, P. Messner, M.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

Schnepp, M. Trunk, P. A. Walker, C. Werle, and P. Winkler, Phys. Rev. X **10**, 031039 (2020).
27. C. Rechatin, J. Faure, X. Davoine, O. Lundh, J. Lim, A. Ben-Ismaïl, F. Burgy, A. Tafzi, A. Lifschitz, E. Lefebvre, and V. Malka, New J. Phys. **12**, 045023 (2010).
28. J. Götzfried, A. Döpp, M. F. Gilljohann, F. M. Foerster, H. Ding, S. Schindler, G. Schilling, A. Buck, L. Veisz, and S. Karsch, Phys. Rev. X **10**, 041015 (2020).
29. M. Kirchen, S. Jalas, P. Messner, P. Winkler, T. Eichner, L. Hübner, T. Hülsenbusch, L. Jeppe, T. Parikh, M. Schnepp, and A. R. Maier, Phys. Rev. Lett. **126**, 174801 (2021).
30. Y. Glinec, J. Faure, L. Le Dain, S. Darbon, T. Hosokai, J. J. Santos, E. Lefebvre, J. P. Rousseau, F. Burgy, B. Mercier, and V. Malka, Phys. Rev. Lett. **94**, 025003 (2005).
31. A. Döpp, E. Guillaume, C. Thaury, A. Lifschitz, F. Sylla, J.-P. Goddet, A. Tafzi, G. Iaquanello, T. Lefrou, P. Rousseau, E. Conejero, C. Ruiz, K. T. Phuoc, and V. Malka, Nucl. Instrum. Methods Phys. Res. Sect. A **830**, 515 (2016).
32. A. Rousse, K. T. Phuoc, R. Shah, A. Pukhov, E. Lefebvre, V. Malka, S. Kiselev, F. Burgy, J.-P. Rousseau, D. Umstadter, and D. Hulin, Phys. Rev. Lett. **93**, 135005 (2004).
33. S. Kneip, C. McGuffey, J. L. Martins, S. F. Martins, C. Bellei, V. Chvykov, F. Dollar, R. Fonseca, C. Huntington, G. Kalintchenko, A. Maksimchuk, S. P. D. Mangles, T. Matsuoka, S. R. Nagel, C. A. J. Palmer, J. Schreiber, K. T. Phuoc, A. G. R. Thomas, V. Yanovsky, L. O. Silva, K. Krushelnick, and Z. Najmudin, Nat. Phys. **6**, 980 (2010).
34. K. T. Phuoc, S. Corde, C. Thaury, V. Malka, A. Tafzi, J. P. Goddet, R. C. Shah, S. Sebban, and A. Rousse, Nat. Photonics **6**, 308 (2012).
35. N. D. Powers, I. Ghebregziabher, G. Golovin, C. Liu, S. Chen, S. Banerjee, J. Zhang, and D. P. Umstadter, Nat. Photonics **8**, 28 (2014).
36. K. Khrennikov, J. Wenz, A. Buck, J. Xu, M. Heigoldt, L. Veisz, and S. Karsch, Phys. Rev. Lett. **114**, 195003 (2015).
37. F. Albert and A. G. R. Thomas, Plasma Phys. Controll. Fusion **58**, 103001 (2016).
38. S. Kneip, C. McGuffey, F. Dollar, M. S. Bloom, V. Chvykov, G. Kalintchenko, K. Krushelnick, A. Maksimchuk, S. P. D. Mangles, T. Matsuoka, Z. Najmudin, C. A. J. Palmer, J. Schreiber, W. Schumaker, A. G. R. Thomas, and V. Yanovsky, Appl. Phys. Lett. **99**, 093701 (2011).
39. S. Fourmaux, S. Corde, K. T. Phuoc, P. Lassonde, G. Lebrun, S. Payeur, F. Martin, S. Sebban, V. Malka, A. Rousse, and J. C. Kieffer, Opt. Lett. **36**, 2426 (2011).
40. J. Wenz, S. Schleede, K. Khrennikov, M. Bech, P. Thibault, M. Heigoldt, F. Pfeiffer, and S. Karsch, Nat. Commun. **6**, 7568 (2015).
41. J. M. Cole, J. C. Wood, N. C. Lopes, K. Poder, R. L. Abel, S. Alatabi, J. S. J. Bryant, A. Jin, S. Kneip, K. Mecseki, D. R. Symes, S. P. D. Mangles, and Z. Najmudin, Sci. Rep. **5**, 13244 (2015).
42. J. M. Cole, D. R. Symes, N. C. Lopes, J. C. Wood, K. Poder, S. Alatabi, S. W. Botchway, P. S. Foster, S. Gratton, S. Johnson, C. Kamperidis, O. Kononenko, M. De Lazzari, C. A. J. Palmer, D. Rusby, J. Sanderson, M. Sandholzer, G. Sarri, Z. Szoke-Kovacs, L. Teboul, J. M. Thompson, J. R. Warwick, H. Westerberg, M. A. Hill, D. P. Norris, S. P. D. Mangles, and Z. Najmudin, Proc. Natl. Acad. Sci. U. S. A. **115**, 6335 (2018).
43. D. Guénot, K. Svendsen, B. Lehnert, H. Ulrich, A. Persson, A. Permogorov, L. Zigan, M. Wensing, O. Lundh, and E. Berrocal, Phys. Rev. Appl. **17**, 064056 (2022).
44. W. Wang, K. Feng, L. Ke, C. Yu, Y. Xu, R. Qi, Y. Chen, Z. Qin, Z. Zhang, M. Fang, J. Liu, K. Jiang, H. Wang, C. Wang, X. Yang, F. Wu, Y. Leng, J. Liu, R. Li, and Z. Xu, Nature **595**, 516 (2021).
45. M. Labat, J. C. Cabadağ, A. Ghaith, A. Irman, A. Berlioux, P. Berteaud, F. Blache, S. Bock, F. Bouvet, F. Briquez, Y.-Y. Chang, S. Corde, A. Debus, C. De Oliveira, J.-P. Duval, Y. Dietrich, M. El Ajjouri, C. Eisenmann, J. Gautier, R. Gebhardt, S. Grams, U. Helbig, C. Herbeaux, N. Hubert, C. Kitegi, O. Kononenko, M. Kuntzsch, M. LaBerge, S. Lê, B. Leluan, A. Loulergue, V. Malka, F. Marteau, M. H. N. Guyen, D. Oumbarek-Espinos, R. Pausch, D. Pereira, T. Püschel, J.-P. Ricaud, P. Rommeluere, E. Roussel, P. Rousseau, S. Schöbel, M. Sebdaoui, K. Steiniger, K. Tavakoli, C. Thaury, P. Ufer, M. Valléau, M. Vandenberghe, J. Vétéran, U. Schramm, and M.-E. Couprie, Nat. Photonics **17**, 150 (2023).
46. C. Emma, J. van Tilborg, F. Albert, L. Labate, J. England, S. Gessner, F. Fiuza, L. Obst-Huebl, A. Zholents, A. Murokh, and J. Rosenzweig, arXiv:2203.09094 (2022).
47. H. Daido, M. Nishiuchi, and A. S. Pirozhkov, Rep. Prog. Phys. **75**, 056401 (2012).
48. A. Macchi, M. Borghesi, and M. Passoni, Rev. Mod. Phys. **85**, 751 (2013).
49. A. P. L. Robinson, M. Zepf, S. Kar, R. G. Evans, and C. Bellei, New J. Phys. **10**, 013021 (2008).
50. A. Henig, S. Steinke, M. Schnürer, T. Sokollik, R. Hörlein, D. Kiefer, D. Jung, J. Schreiber, B. M. Hegelich, X. Q. Yan, J. Meyer-ter-Vehn, T. Tajima, P. V. Nickles, W. Sandner, and D. Habs, Phys. Rev. Lett. **103**, 245003 (2009).
51. R. A. C. Fraga, A. Kalinin, M. Khnel, D. C. Hochhaus, A. Schottelius, J. Polz, M. C. Kaluza, P. Neumayer, and R. E. Grisenti, Rev. Sci. Instrum. **83**, 025102 (2012).
52. M. Gauthier, J. B. Kim, C. B. Curry, B. Aurand, E. J. Gamboa, S. Göde, C. Goyon, A. Hazi, S. Kerr, A. Pak, A. Propp, B. Ramakrishna, J. Ruby, O. Willi, G. J. Williams, C. Rödel, and S. H. Glenzer, Rev. Sci. Instrum. **87**, 11D827 (2016).
53. S. D. Kraft, L. Obst, J. Metzkes-Ng, H. P. Schlenvoigt, K. Zeil, S. Michaux, D. Chatain, J. P. Perin, S. N. Chen, J. Fuchs, M. Gauthier, T. E. Cowan, and U. Schramm, Plasma Phys. Controll. Fusion **60**, 044010 (2018).
54. M. Rehwald, S. Assenbaum, C. Bernert, C. B. Curry, M. Gauthier, S. H. Glenzer, S. Göde, C. Schoenwaelder, U. Schramm, F. Treffert, and K. Zeil, J. Phys. Conf. Ser. **2420**, 012034 (2023).
55. D. W. Schumacher, P. L. Poole, C. Willis, G. E. Cochran, R. Daskalova, J. Purcell, and R. Heery, J. Instrum. **12**, C04023 (2017).
56. E. I. Moses, Nucl. Fusion **49**, 104022 (2009).
57. R. Betti and O. A. Hurricane, Nat. Phys. **12**, 435 (2016).
58. M. Murakami and J. Meyer-ter-Vehn, Nucl. Fusion **31**, 1315 (1991).
59. R. S. Craxton, K. S. Anderson, T. R. Boehly, V. N. Goncharov, D. R. Harding, J. P. Knauer, R. L. McCrory, P. W. McKenty, D. D. Meyerhofer, J. F. Myatt, A. J. Schmitt, J. D. Sethian, R. W. Short, S. Skupsky, W. Theobald, W. L. Kruer, K. Tanaka, R. Betti, T. J. B. Collins, J. A. Delettrez, S. X. Hu, J. A. Marozas, A. V. Maximov, D. T. Michel, P. B. Radha, S. P. Regan, T. C. Sangster, W. Seka, A. A. Solodov, J. M. Soures, C. Stoeckl, and J. D. Zuegel, Phys. Plasmas **22**, 110501 (2015).
60. R. Kodama, P. A. Norreys, K. Mima, A. E. Dangor, R. G. Evans, H. Fujita, Y. Kitagawa, K. Krushelnick, T. Miyakoshi, N. Miyanaga, T. Norimatsu, S. J. Rose, T. Shozaki, K. Shigemori, A. Sunahara, M. Tampo, K. A. Tanaka, Y. Toyama, T. Yamanaka, and M. Zepf, Nature **412**, 798 (2001).
61. R. Betti, C. D. Zhou, K. S. Anderson, L. J. Perkins, W. Theobald, and A. A. Solodov, Phys. Rev. Lett. **98**, 155001 (2007).
62. A. B. Zylstra, O. A. Hurricane, D. A. Callahan, A. L. Kritcher, J. E. Ralph, H. F. Robey, J. S. Ross, C. V. Young, K.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

L. Baker, D. T. Casey, T. Döppner, L. Divol, M. Hohenberger, S. Le Pape, A. Pak, P. K. Patel, R. Tommasini, S. J. Ali, P. A. Amendt, L. J. Atherton, B. Bachmann, D. Bailey, L. R. Benedetti, L. Berzak Hopkins, R. Betti, S. D. Bhandarkar, J. Biener, R. M. Bionta, N. W. Birge, E. J. Bond, D. K. Bradley, T. Braun, T. M. Briggs, M. W. Bruhn, P. M. Celliers, B. Chang, T. Chapman, H. Chen, C. Choate, A. R. Christopherson, D. S. Clark, J. W. Crippen, E. L. Dewald, T. R. Dittrich, M. J. Edwards, W. A. Farmer, J. E. Field, D. Fittinghoff, J. Frenje, J. Gaffney, M. Gatu Johnson, S. H. Glenzer, G. P. Grim, S. Haan, K. D. Hahn, G. N. Hall, B. A. Hammel, J. Harte, E. Hartouni, J. E. Heebner, V. J. Hernandez, H. Herrmann, M. C. Herrmann, D. E. Hinkel, D. D. Ho, J. P. Holder, W. W. Hsing, H. Huang, K. D. Humbird, N. Izumi, L. C. Jarrott, J. Jeet, O. Jones, G. D. Kerbel, S. M. Kerr, S. F. Khan, J. Kilkenny, Y. Kim, H. G. Kleinrath, V. G. Kleinrath, C. Kong, J. M. Koning, J. J. Kroll, M. K. G. Kruse, B. Kustowski, O. L. Landen, S. Langer, D. Larson, N. C. Lemos, J. D. Lindl, T. Ma, M. J. MacDonald, B. J. MacGowan, A. J. Mackinnon, S. A. MacLaren, A. G. MacPhee, M. M. Marinak, D. A. Mariscal, E. V. Marley, L. Masse, K. Meaney, N. B. Meezan, P. A. Michel, M. Millot, J. L. Milovich, J. D. Moody, A. S. Moore, J. W. Morton, T. Murphy, K. Newman, J.-M. G. Di Nicola, A. Nikroo, R. Nora, M. V. Patel, L. J. Pelz, J. L. Peterson, Y. Ping, B. B. Pollock, M. Ratledge, N. G. Rice, H. Rinderknecht, M. Rosen, M. S. Rubery, J. D. Salmonson, J. Sater, S. Schiaffino, D. J. Schlossberg, M. B. Schneider, C. R. Schroeder, H. A. Scott, S. M. Sepke, K. Sequoia, M. W. Sherlock, S. Shin, V. A. Smalyuk, B. K. Spears, P. T. Springer, M. Stadermann, S. Stoupin, D. J. Strozzi, L. J. Suter, C. A. Thomas, R. P. J. Town, E. R. Tubman, C. Trosseille, P. L. Volegov, C. R. Weber, K. Widmann, C. Wild, C. H. Wilde, B. M. Van Wonterghem, D. T. Woods, B. N. Woodworth, M. Yamaguchi, S. T. Yang, and G. B. Zimmerman, Nature **601**, 542 (2022).
63. G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová, Rev. Mod. Phys. **91**, 045002 (2019).
64. V. Dunjko and H. J. Briegel, Rep. Prog. Phys. **81**, 074001 (2018).
65. K. T. Schütt, S. Chmiela, O. A. von Lilienfeld, A. Tkatchenko, K. Tsuda, and K.-R. Müller, *Machine Learning Meets Quantum Physics*, Lecture Notes in Physics (Springer, 2020).
66. A. Radovic, M. Williams, D. Rousseau, M. Kagan, D. Bonacorsi, A. Himmel, A. Aurisano, K. Terao, and T. Wongjirad, Nature **560**, 41 (2018).
67. E. Bedolla, L. C. Padierna, and R. Castaneda-Priego, J. Phys. Condens. Matter **33**, 053001 (2020).
68. J. M. Ede, Mach. Sci. Technol. **2**, 011004 (2021).
69. J. N. Kutz, J. Fluid Mech. **814**, 1 (2017).
70. W. Leemans, "Report of Workshop on Laser Technology for k-BELLA and Beyond," https://www2.lbl.gov/LBL-Programs/atap/Report_Workshop_k-BELLA_laser_tech_final.pdf (2017).
71. I. Prencipe, J. Fuchs, S. Pascarelli, D. W. Schumacher, R. B. Stephens, N. B. Alexander, R. Briggs, M. Büscher, M. O. Cernaianu, A. Choukourov, M. De Marco, A. Erbe, J. Fassbender, G. Fiquet, P. Fitzsimmons, C. Gheorghiu, J. Hund, L. G. Huang, M. Harmand, N. J. Hartley, A. Irman, T. Kluge, Z. Konopkova, S. Kraft, D. Kraus, V. Leca, D. Margarone, J. Metzkes, K. Nagai, W. Nazarov, P. Lutoslawski, D. Papp, M. Passoni, A. Pelka, J. P. Perin, J. Schulz, M. Smid, C. Spindloe, S. Steinke, R. Torchio, C. Vass, T. Wiste, R. Zaffino, K. Zeil, T. Tschentscher, U. Schramm, and T. E. Cowa, High Power Laser Sci. Eng. **5**, e17 (2017).
72. K. M. George, J. T. Morrison, S. Feister, G. K. Ngirmang, J. R. Smith, A. J. Klim, J. Snyder, D. Austin, W. Erbsen, K. D. Frische, J. Nees, C. Orban, E. A. Chowdhury, and W. M. Roquemore, High Power Laser Sci. Eng. **7**, e50 (2019).
73. T. Chagovets, S. Stanček, L. Giuffrida, A. Velyhan, M. Tryus, F. Grepl, V. Istokskaia, V. Kantarelou, T. Wiste, J. C. H. Martin, F. Schillaci, and D. Margarone, Appl. Sci. **11**, 1680 (2021).
74. F. P. Condamine, N. Jourdain, J.-C. Hernandez, M. Taylor, H. Bohlin, A. Fajstavr, T. M. Jeong, D. Kumar, T. Laštovička, O. Renner, and S. Weber, Rev. Sci. Instrum. **92**, 063504 (2021).
75. K. Nakamura, H.-S. Mao, A. J. Gonsalves, H. Vincenti, D. E. Mittelberger, J. Daniels, A. Magana, C. Toth, and W. P. Leemans, IEEE J. Quantum Electron. **53**, 1200121 (2017).
76. E. Sistrunk, T. Spinka, A. Bayramian, S. Betts, R. Bopp, S. Buck, K. Charron, J. Cupal, R. Deri, M. Drouin, A. Erlandson, E. S. Fulkerson, J. Horner, J. Horacek, J. Jarboe, K. Kasl, D. Kim, E. Koh, L. Koubikova, R. Lanning, W. Maranville, C. Marshall, D. Mason, J. Menapace, P. Miller, P. Mazurek, A. Naylon, J. Novak, D. Peceli, P. Rosso, K. Schaffers, D. Smith, J. Stanley, R. Steele, S. Telford, J. Thoma, D. VanBlarcom, J. Weiss, P. Wegner, B. Rus, and C. Haefne, in *CLEO: Science and Innovations* (Optica Publishing Group, 2017), paper STh1L.2.
77. L. Roso, EPJ Web Conf. **167**, 01001 (2018).
78. J. Pilar, M. De Vido, M. Divoky, P. Mason, M. Hanus, K. Ertel, P. Navratil, T. Butcher, O. Slezak, S. Banerjee, J. Phillips, J. Smith, A. Lucianetti, C. Hernandez-Gomez, C. Edwards, J. Collier, and T. Mocek, Proc. SPIE **10511**, 105110X (2018).
79. N. Jourdain, U. Chaulagain, M. Havlík, D. Kramer, D. Kumar, I. Majerová, V. T. Tikhonchuk, G. Korn, and S. Weber, Matter Radiat. Extremes **6**, 015401 (2021).
80. B. K. Spears, J. Brase, P.-T. Bremer, B. Chen, J. Field, J. Gaffney, M. Kruse, S. Langer, K. Lewis, R. Nora, J. L. Peterson, J. Thiagarajan, B. Van Essen, and K. Humbird, Phys. Plasmas **25**, 080901 (2018).
81. R. Anirudh, R. Archibald, M. S. Asif, M. M. Becker, S. Benkadda, P.-T. Bremer, R. H. S. Budé, C. S. Chang, L. Chen, R. M. Churchill, J. Citrin, J. A. Gaffney, A. Gainaru, W. Gekelman, T. Gibbs, S. Hamaguchi, C. Hill, K. Humbird, S. Jalas, S. Kawaguchi, G.-H. Kim, M. Kirchen, S. Klasky, J. L. Kline, K. Krushelnick, B. Kustowski, G. Lapenta, W. Li, T. Ma, N. J. Mason, A. Mesbah, C. Michoski, T. Munson, I. Murakami, H. N. Najm, K. E. J. Olofsson, S. Park, J. L. Peterson, M. Probst, D. Pugmire, B. Sammuli, K. Sawlani, A. Scheinker, D. P. Schissel, R. J. Shalloo, J. Shinagawa, J. Seong, B. K. Spears, J. Tennyson, J. Thiagarajan, C. M. Ticoş, J. Trieschmann, J. van Dijk, B. Van Essen, P. Ventzek, H. Wang, J. T. L. Wang, Z. Wang, K. Wende, X. Xu, H. Yamada, T. Yokoyama, and X. Zhang, arXiv:2205.15832 (2022).
82. M. Kambara, S. Kawaguchi, H. J. Lee, K. Ikuse, S. Hamaguchi, T. Ohmori, and K. Ishikawa, Jpn. J. Appl. Phys. **62**, SA0803 (2022).
83. G. Genty, L. Salmela, J. M. Dudley, D. Brunner, A. Kokhanovskiy, S. Kobtsev, and S. K. Turitsyn, Nat. Photonics **15**, 91 (2021).
84. P. W. Hatfield, J. A. Gaffney, G. J. Anderson, S. Ali, L. Antonelli, S. B. du Pree, J. Citrin, M. Fajardo, P. Knapp, B. Kettle, B. Kustowski, M. J. MacDonald, D. Mariscal, M. E. Martin, T. Nagayama, C. A. J. Palmer, J. L. Peterson, S. Rose, J. J. Ruby, C. Shneider, M. J. V. Streeter, W. Trickey, and B. Williams, Nature **593**, 351 (2021).
85. A. Rasheed, O. San, and T. Kvamsdal, IEEE Access **8**, 21980 (2020).
86. D. Anderson and K. Burnham, *Model Selection and Multi-Model Inference*, 2nd ed. (Springer, 2002).

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

87. D. J. C. MacKay, NATO ASI Ser. F: Comput. Syst. Sci. **168**, 133 (1998).
88. C. E. Rasmussen, in *Advanced Lectures on Machine Learning* (Springer, 2003), p. 63.
89. M. G. Genton, J. Mach. Learn. Res. **2**, 299 (2001).
90. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees* (Routledge, New York, NY, 2017).
91. Y. Freund and R. E. Schapire, J. Jpn. Soc. Artif. Intell. **14**, 771 (1999).
92. J. H. Friedman, Ann. Stat. **29**, 1189 (2001).
93. J. H. Friedman, Comput. Stat. Data Analysis **38**, 367 (2002).
94. K. D. Humbird, J. L. Peterson, and R. G. McClarren, IEEE Trans. Neural Networks Learn. Syst. **30**, 1286 (2018).
95. K. D. Humbird, J. L. Peterson, B. K. Spears, and R. G. McClarren, IEEE Trans. Plasma Sci. **48**, 61 (2019).
96. A. Hsu, B. Cheng, and P. A. Bradley, Phys. Plasmas **27**, 012703 (2020).
97. G. Kluth, K. D. Humbird, B. K. Spears, J. L. Peterson, H. A. Scott, M. V. Patel, J. Koning, M. Marinak, L. Divol, and C. V. Young, Phys. Plasmas **27**, 052707 (2020).
98. J. Lin, Q. Qian, J. Murphy, A. Hsu, A. Hero, Y. Ma, A. G. R. Thomas, and K. Krushelnick, Phys. Plasmas **28**, 083102 (2021).
99. B. J. Wythoff, Chem. Intell. Lab. Syst. **18**, 115 (1993).
100. D. P. Kingma and J. Ba, arXiv:1412.6980 (2017).
101. A. Kristiadi, M. Hein, and P. Hennig, arXiv:2002.10118 (2020).
102. S. Wager, S. Wang, and P. S. Liang, arXiv:1307.1493 (2013).
103. J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, M. Shahzad, W. Yang, R. Bamler, and X. X. Zhu, arXiv:2107.03342 (2022).
104. G. Montavon, W. Samek, and K.-R. Müller, Digital Sign. Process. **73**, 1 (2018).
105. M. Huh, P. Agrawal, and A. A. Efros, arXiv:1608.08614 (2016).
106. A. Gonoskov, E. Wallin, A. Polovinkin, and I. Meyerov, Sci. Rep. **9**, 7043 (2019).
107. Y. Rodimkov, S. Bhadoria, V. Volokitin, E. Efimenko, A. Polovinkin, T. Blackburn, M. Marklund, A. Gonoskov, and I. Meyerov, Sensors **21**, 6982 (2021).
108. B. Z. Djordjević, A. J. Kemp, J. Kim, R. A. Simpson, S. C. Wilks, T. Ma, and D. A. Mariscal, Phys. Plasmas **28**, 043105 (2021).
109. R. A. Watt, "Monte Carlo modelling of QED Interactions in laser-plasma experiments," PhD. Thesis (Imperial College London, 2021).
110. R.G. McClarren, I. L. Tregillis, T. J. Urbatsch, and E. S. Dodd, Phys. Lett. A **396**, 127243 (2021).
111. R. A. Simpson, D. Mariscal, G. J. Williams, G. G. Scott, E. Grace, and T. Ma, Rev. Sci. Instrum. **92**, 075101 (2021).
112. M. J. V. Streeter, C. Colgan, C. C. Cobo, C. Arran, E. E. Los, R. Watt, N. Bourgeois, L. Calvin, J. Carderelli, N. Cavanagh, S. J. D. Dann, R. Fitzgarrald, E. Gerstmayr, A. S. Joglekar, B. Kettle, P. Mckenna, C. D. Murphy, Z. Najmudin, P. Parsons, Q. Qian, P. P. Rajeev, C. P. Ridgers, D. R. Symes, A. G. R. Thomas, G. Sarri, and S. P. D. Mangles, High Power Laser Sci. Eng. **11**, e9 (2023).
113. T. Wu and M. Tegmark, Phys. Rev. E **100**, 033311 (2019).
114. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf (2018).
115. S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, arXiv:2303.12712 (2023).
116. M. Schmidt and H. Lipson, Science **324**, 81 (2009).
117. S. L. Brunton, J. L. Proctor, and J. N. Kutz, Proc. Natl. Acad. Sci. U. S. A. **113**, 3932 (2016).
118. G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, Nat. Rev. Phys. **3**, 422 (2021).
119. M. Raissi, P. Perdikaris, and G. E. Karniadakis, arXiv:1711.10561 (2017).
120. M. Raissi, P. Perdikaris, and G. E. Karniadakis, arXiv:1711.10566 (2017).
121. M. Raissi, P. Perdikaris, and G. E. Karniadakis, J. Comput. Phys. **378**, 686 (2019).
122. M. Raissi, A. Yazdani, and G. E. Karniadakis, Science **367**, 1026 (2020).
123. Z. Long, Y. Lu, X. Ma, and B. Dong, arXiv:1710.09668 (2018).
124. S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, J. Sci. Comput. **92**, 88 (2022).
125. S. Kawaguchi and T. Murakami, Jpn. J. Appl. Phys. **61**, 086002 (2022).
126. P. Stiller, F. Bethke, M. Böhme, R. Pausch, S. Torge, A. Debus, J. Vorberger, M. Bussmann, and N. Hoffmann, arXiv:2009.03730 (2020).
127. J. J. F. Commandeur and S. J. Koopman, *An Introduction to State Space Time Series Analysis* (Oxford University Press, 2007).
128. W. Zucchini and I. L. MacDonald, *Hidden Markov Models for Time Series: An Introduction Using R* (CRC Press, 2009).
129. R. E. Kalman, J. Basic Eng. **82**, 35 (1960).
130. F. Auger, M. Hilairet, J. M. Guerrero, E. Monmasson, T. Orlowska-Kowalska, and S. Katsura, IEEE Trans. Indus. Electron. **60**, 5458 (2013).
131. L. A. Poyneer and J.-P. Véran, J. Opt. Soc. Am. A **27**, A223 (2010).
132. A. Ushakov, P. Echevarria, and A. Neumann, in *8th International Particle Accelerator Conference* (2017), p. 3938.
133. G. Welch, and G. Bishop, "An introduction to the Kalman filter," https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf (1995).
134. B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman Filter: Particle Filters for Tracking Applications* (Artech House, 2003).
135. H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, Data Mining Knowledge Discovery **33**, 917 (2019).
136. B. Lim and S. Zohren, Philos. Trans. R. Soc. A **379**, 20200209 (2021).
137. S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," Diploma (Technische Universität München, 1991).
138. S. Hochreiter, Int. J. Uncertainty Fuzziness Knowl. Syst. **6**, 107 (1998).
139. S. Hochreiter and J. Schmidhuber, Neural Comput. **9**, 1735 (1997).
140. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, Adv. Neural Inform. Process. Syst. **30** (2017).
141. B. Lim, S. Ö. Ark, N. Loeff, and T. Pfister, Int. J. Forecasting **37**, 1748 (2021).
142. Z. Wang, W. Yan, and T. Oates, in *2017 International Joint Conference on Neural Networks (IJCNN)* (IEEE, 2017), p. 1578.
143. T. D. Bui, C. Nguyen, and R. E. Turner, arXiv:1705.07131 (2017).
144. I. Žliobaitė, M. Pechenizkiy, and J. Gama, in *Big Data Analysis: New Algorithms for a New Society* (Springer, 2016), p. 91.

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

145. K. Cassou, in *LPA Online Workshop on Control Systems and Machine* Learning (2022).
146. N. Weiße, L. Doyle, J. Gebhard, F. Balling, F. Schweiger, F. Haberstroh, L. D. Geulig, J. Lin, F. Irshad, J. Esslinger, S. Gerlach, M. Gilljohann, V. Vaidyanathan, D. Siebert, A. Münzer, G. Schilling, J. Schreiber, P. G. Thirolf, S. Karsch, and A. Döpp, High Power Laser Sci. Eng. **11**, e44 (2023).
147. T. Ma, D. Mariscal, R. Anirudh, T. Bremer, B. Z. Djordjevic, T. Galvin, E. Grace, S. Herriot, S. Jacobs, B. Kailkhura, R. Hollinger, J. Kim, S. Liu, J. Ludwig, D. Neely, J. J. Rocca, G. G. Scott, R. A. Simpson, B. S. Spears, T. S. Spinka, K. Swanson, J. J. Thiagarajan, B. Van Essen, S. Wang, S. C. Wilks, G. J. Williams, J. Zhang, M. C. Herrmann, and C. Haefner, Plasma Phys. Controll. Fusion **63**, 104003 (2021).
148. J. C. A. Barata and M. S. Hussein, Braz. J. Phys. **42**, 146 (2012).
149. J. M. Bioucas-Dias and M. A. T. Figueiredo, IEEE Trans. Image Process. **16**, 2992 (2007).
150. A. Beck and M. Teboulle, SIAM J. Imaging Sci. **2**, 183 (2009).
151. J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., Springer Series in Operations Research and Financial Engineering (Springer, 2007).
152. R. K. Tyson and B. W. Frazier, *Principles of Adaptive Optics* (CRC Press, 2022).
153. A. Tarantola, *Inverse Problem Theory and Methods for Model Parameter Estimation* (SIAM, 2005).
154. C. J. Geyer, "Markov chain Monte Carlo maximum likelihood," https://www.stat.umn.edu/geyer/f05/8931/c.pdf (1991).
155. A. P. Dempster, N. M. Laird, and D. B. Rubin, J. R. Stat. Soc. Ser. B **39**, 1 (1977).
156. H. Zhang, J. Wang, D. Zeng, X. Tao, and J. Ma, Med. Phys. **45**, e886 (2018).
157. K. Cranmer, J. Brehmer, and G. Louppe, Proc. Natl. Acad. Sci. U. S. A. **117**, 30055 (2020).
158. S. A. Sisson, Y. Fan, and M. Beaumont, *Handbook of Approximate Bayesian Computation* (CRC Press, 2018).
159. W. R. Gilks, S. Richardson, and D. Spiegelhalter, *Markov Chain Monte Carlo in Practice* (CRC Press, 1995).
160. M. U. Gutmann and J. Corander, J. Mach. Learn. Res. **17**, 1 (2016).
161. J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez, Phys. Rev. D **98**, 052004 (2018).
162. A. Döpp, L. Hehn, J. Götzfried, J. Wenz, M. Gilljohann, H. Ding, S. Schindler, F. Pfeiffer, and S. Karsch, Optica **5**, 199 (2018).
163. J. Götzfried, A. Döpp, M. Gilljohann, H. Ding, S. Schindler, J. Wenz, L. Hehn, F. Pfeiffer, and S. Karsch, Nucl. Instrum. Methods Phys. Res. Sect. A **909**, 286 (2018).
164. E. J. Candes and M. B. Wakin, IEEE Sig. Process. Mag. **25**, 21 (2008).
165. G. Oliveri, M. Salucci, N. Anselmi, and A. Massa, IEEE Antennas Propag. Mag. **59**, 34 (2017).
166. X. Yuan, D. J. Brady, and A. K. Katsaggelos, IEEE Sig. Process. Mag. **38**, 65 (2021).
167. D. L. Donoho, IEEE Trans. Inform. Theory **52**, 1289 (2006).
168. E. J. Candès, M. B. Wakin, and S. P. Boyd, J. Fourier Anal. Appl. **14**, 877 (2008).
169. A. Kohlenberg, J. Appl. Phys. **24**, 1432 (1953).
170. J. Liang, P. Wang, L. Zhu, and L. V. Wang, in *OSA Imaging and Applied Optics Congress* (Optica Publishing Group, 2021), paper 3Tu4A.1.
171. Y. Huang, S. Jiang, H. Li, Q. Wang, and L. Chen, Comput. Phys. Commun. **185**, 459 (2014).
172. M. Kassubeck, S. Wenger, C. Jennings, M. R. Gomez, E. Harding, J. Schwarz, and M. Magnor, in *IS&T International Symposium on Electronic Imaging* (Society for Imaging Science and Technology, 2018), p. 1331.
173. Y. Ma, J. Hua, D. Liu, Y. He, T. Zhang, J. Chen, F. Yang, X. Ning, Z. Yang, J. Zhang, C.-H. Pai, Y. Gu, and W. Lu, Matter Radiat. Extremes **5**, 064401 (2020).
174. H. Yu and G. Wang, Phys. Med. Biol. **54**, 2791 (2009).
175. S. Arridge, P. Maass, O. Öktem, and C.-B. Schönlieb, Acta Numer. **28**, 1 (2019).
176. B. Bermeitinger, T. Hrycej, and S. Handschuh, arXiv:1906.11755 (2019).
177. I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, arXiv:1406.2661 (2014).
178. H. Thanh-Tung and T. Tran, arXiv:1807.04015 (2020).
179. P. Peng, S. Jalali, and X. Yuan, arXiv:1901.05045 (2019).
180. O. Ronneberger, P. Fischer, and T. Brox, arXiv:1505.04597 (2015).
181. L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother, and U. Köthe, arXiv:1808.04730 (2018).
182. D. Rezende and S. Mohamed, arXiv:1505.05770 (2016).
183. L. Dinh, D. Krueger, and Y. Bengio, arXiv:1410.8516 (2014).
184. A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, J. Mach. Learn. Res. **13**, 723 (2012).
185. L. Ardizzone, C. Lüth, J. Kruse, C. Rother, and U. Köthe, arXiv:1907.02392 (2019).
186. M. A. Krumbügel, C. L. Ladera, K. W. DeLong, D. N. Fittinghoff, J. N. Sweetser, and R. Trebino, Opt. Lett. **21**, 143 (1996).
187. T. Zahavy, A. Dikopoltsev, D. Moss, G. I. Haham, O. Cohen, S. Mannor, and M. Segev, Optica **5**, 666 (2018).
188. L. Hu, S. Hu, W. Gong, and K. Si, Opt. Lett. **45**, 3741 (2020).
189. F. Bethke, R. Pausch, P. Stiller, A. Debus, M. Bussmann, and N. Hoffmann, arXiv:2106.00432 (2021).
190. V. Monga, Y. Li, and Y. C. Eldar, arXiv:1912.10557 (2019).
191. H. Li, J. Schwab, S. Antholzer, and M. Haltmeier, Inverse Probl. **36**, 065005 (2020).
192. S. Howard, J. Esslinger, R. H.W. Wang, P. Norreys, and A. Döpp, High Power Laser Sci. Eng. **11**, e32 (2023).
193. M. E. Gehm, R. John, D. J. Brady, R. M. Willett, and T. J. Schulz, Opt. Express **15**, 14013 (2007).
194. Z.-H. He, B. Hou, V. Lebailly, J. A. Nees, K. Krushelnick, and A. G. R. Thomas, Nat. Commun. **6**, 7156 (2015).
195. S. J. D. Dann, C. D. Baird, N. Bourgeois, O. Chekhlov, S. Eardley, C. D. Gregory, J.-N. Gruse, J. Hah, D. Hazra, S. J. Hawkes, C. J. Hooker, K. Krushelnick, S. P. D. Mangles, V. A. Marshall, C. D. Murphy, Z. Najmudin, J. A. Nees, J. Osterhoff, B. Parry, P. Pourmoussavi, S. V. Rahul, P. P. Rajeev, S. Rozario, J. D. E. Scott, R. A. Smith, E. Springate, Y. Tang, S. Tata, A. G. R. Thomas, C. Thornton, D. R. Symes, and M. J. V. Streeter. Phys. Rev. Accel. Beams **22**, 041303 (2019).
196. R. J. Shalloo, S. J. D. Dann, J.-N. Gruse, C. I. D. Underwood, A. F. Antoine, C. Arran, M. Backhouse, C. D. Baird, M. D. Balcazar, N. Bourgeois, J. A. Cardarelli, P. Hatfield, J. Kang, K. Krushelnick, S. P. D. Mangles, C. D. Murphy, N. Lu, J. Osterhoff, K. Põder, P. P. Rajeev, C. P. Ridgers, S. Rozario, M. P. Selwood, A. J. Shahani, D. R. Symes, A. G. R. Thomas, C. Thornton, Z. Najmudin, and M. J. V. Streeter, Nat. Commun. **11**, 6355 (2020).
197. S. Jalas, M. Kirchen, P. Messner, P. Winkler, L. Hübner, J. Dirkwinkel, M. Schnepp, R. Lehe, and A. R. Maier, Phys. Rev. Lett. **126**, 104801 (2021).
198. F. Irshad, S. Karsch, and A. Döpp, Phys. Rev. Res. **5**, 013063 (2023).

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

199. F. Irshad, S. Karsch, and A. Döpp, arXiv:2112.13901 (2021).
200. L. C. W. Dixon and G. P. Szego, Toward Global Optim. **2**, 1 (1978).
201. C. Currin, T. Mitchell, M. Morris, and D. Ylvisaker, J. Amer. Stat. Assoc. **86**, 953 (1991).
202. J. Bergstra and Y. Bengio, J. Mach. Learn. Res. **13**, 281 (2012).
203. T. Kollig and A. Keller, Comput. Graph. Forum **21**, 557 (2002).
204. J. A. Nelder and R. Mead, Comput. J. **7**, 308 (1965).
205. R. Storn and K. Price, J. Global Optim. **11**, 341 (1997).
206. K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, IEEE Trans. Evolution. Comput. **6**, 182 (2001).
207. J. Horn, N. Nafpliotis, and D. E. Goldberg, in *Proceedings of the First IEEE Conference on Evolutionary Computation and IEEE World Congress on Computational Intelligence* (IEEE, 1994), p. 82.
208. R. Bartels, S. Backus, E. Zeek, L. Misoguti, G. Vdovin, I. P. Christov, M. M. Murnane, and H. C. Kapteyn, Nature **406**, 164 (2000).
209. D. Yoshitomi, J. Nees, N. Miyamoto, T. Sekikawa, T. Kanai, G. Mourou, and S. Watanabe, Appl. Phys. B **78**, 275 (2004).
210. A. S. Moore, K. J. Mendham, D. R. Symes, J. S. Robinson, E. Springate, M. B. Mason, R. A. Smith, J. W. G. Tisch, and J. P. Marangos, Appl. Phys. B **80**, 101 (2005).
211. S. Zamith, T. Martchenko, Y. Ni, S. A. Aseyev, H. G. Muller, and M. J. J. Vrakking, Phys. Rev. A **70**, 011201 (2004).
212. T. Nayuki, T. Fujii, Y. Oishi, K. Takano, X. Wang, A. A. Andreev, K. Nemoto, and K. Ueda, Rev. Sci. Instrum. **76**, 073305 (2005).
213. Z.-H. He, B. Hou, G. Gao, V. Lebailly, J. A. Nees, R. Clarke, K. Krushelnick, and A. G. R. Thomas, Phys. Plasmas **22**, 056704 (2015).
214. J. Lin, Y. Ma, R. Schwartz, D. Woodbury, J. A. Nees, M. Mathis, A. G. R. Thomas, K. Krushelnick, and H. Milchberg, Opt. Express **27**, 10912 (2019).
215. J. Hah, W. Jiang, Z. H. He, J. A. Nees, B. Hou, A. G. R. Thomas, and K. Krushelnick, Opt. Express **25**, 17271 (2017).
216. J. Lin, J. H. Easter, K. Krushelnick, M. Mathis, J. Dong, A. G. R. Thomas, and J. Nees, Opt. Commun. **421**, 79 (2018).
217. M. Noaman-ul-Haq, T. Sokollik, H. Ahmed, J. Braenzel, L. Ehrentraut, M. Mirzaie, L.-L. Yu, Z. M. Sheng, L. M. Chen, M. Schnürer, and J. Zhang, Nucl. Instrum. Methods Phys. Res. Sect. A **883**, 191 (2018).
218. L. A. Finney, J. Lin, P. J. Skrodzki, M. Burger, J. Nees, K. Krushelnick, and I. Jovanovic, Opt. Commun. **490**, 126902 (2021).
219. A. Englesbe, J. Lin, J. Nees, A. Lucero, K. Krushelnick, and A. Schmitt-Sody, Appl. Opt. **60**, G113 (2021).
220. M. J. V. Streeter, S. J. D. Dann, J. D. E. Scott, C. D. Baird, C. D. Murphy, S. Eardley, R. A. Smith, S. Rozario, J.-N. Gruse, S. P. D. Mangles, Z. Najmudin, S. Tata, M. Krishnamurthy, S. V. Rahul, D. Hazra, P. Pourmoussavi, J. Osterhoff, J. Hah, N. Bourgeois, C. Thornton, C. D. Gregory, C. J. Hooker, O. Chekhlov, S. J. Hawkes, B. Parry, V. A. Marshall, Y. Tang, E. Springate, P. P. Rajeev, A. G. R. Thomas, and D. R. Symes, Appl. Phys. Lett. **112**, 244101 (2018).
221. D. Peceli and P. Mazurek, in *LPA Online Workshop on Control Systems and Machine* Learning (2022).
222. J. R. Smith, C. Orban, J. T. Morrison, K. M. George, G. K. Ngirmang, E. A. Chowdhury, and W. M. Roquemore, New J. Phys. **22**, 103067 (2020).
223. B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, Proc. IEEE **104**, 148 (2015).
224. P. I. Frazier, arXiv:1807.02811 (2018).
225. D. R. Jones, M. Schonlau, and W. J. Welch, J. Global Optim. **13**, 455 (1998).
226. P. Frazier, W. Powell, and S. Dayanik, INFORMS J. Comput. **21**, 599 (2009).
227. P. Hennig and C. J. Schuler, J. Mach. Learn. Res. **13**, 1809 (2012).
228. K. Yang, M. Emmerich, A. Deutz, and T. Bäck, Swarm Evolut. Comput. **44**, 945 (2019).
229. A. Marco, F. Berkenkamp, P. Hennig, A. P. Schoellig, A. Krause, S. Schaal, and S. Trimpe, in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2017), p. 1557.
230. R. Lehe, in *GPU Technology Conference* (2017).
231. E. J. Dolier, M. King, R. Wilson, R. J. Gray, and P. McKenna, New J. Phys. **24**, 073025 (2022).
232. B. Loughran, M. J. V. Streeter, H. Ahmed, S. Astbury, M. Balcazar, M. Borghesi, N. Bourgeois, C. B. Curry, S. J. D. Dann, S. DiIorio, N. P. Dover, T. Dzelzanis, O. C. Ettlinger, M. Gauthier, L. Giuffrida, G. D. Glenn, S. H. Glenzer, J. S. Green, R. J. Gray, G. S. Hicks, C. Hyland, V. Istokskaia, M. King, D. Margarone, O. McCusker, P. McKenna, Z. Najmudin, C. Parisuaña, P. Parsons, C. Spindloe, D. R. Symes, A. G. R. Thomas, F. Treffert, N. Xu, and C. A. J. Palmer, arXiv:2303.00823 (2023).
233. Á. F. Pousa, S. T. P. Hudson, A. Huebl, S. Jalas, M. Kirchen, J. M. Larson, R. Lehé, A. M. de la Ossa, M. Thévenet, and J.-L. Vay, in *Proceedings of the 13th International Particle Accelerator Conference* (JACoW Publishing, 2022), p. 1761.
234. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, 2018).
235. K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, IEEE Signal Process. Mag. **34**, 26 (2017).
236. Y. Li, arXiv:1810.06339 (2018).
237. I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, IEEE Trans. Syst. Man, Cybernet. C **42**, 1291 (2012).
238. R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, in *Proceedings of the 12th International Conference on Neural Information Processing Systems* (ACM, 1999), p. 1057.
239. V. Kain, S. Hirlander, B. Goddard, F. M. Velotti, G. Z. D. Porta, N. Bruchon, and G. Valentino, Phys. Rev. Accel. Beams **23**, 124801 (2020).
240. Y. Gao, J. Chen, T. Robertazzi, and K. A. Brown. Phys. Rev. Accel. Beams **22**, 014601 (2019).
241. N. Bruchon, G. Fenu, G. Gaio, M. Lonza, F. H. O'Shea, F. A. Pellegrino, and E. Salvato, Electronics **9**, 781 (2020).
242. F. H. O'Shea, N. Bruchon, and G. Gaio. Phys. Rev. Accel. Beams **23**, 122802 (2020).
243. J. S. John, C. Herwig, D. Kafkes, J. Mitrevski, W. A. Pellico, G.N. Perdue, A. Quintero-Parra, B. A. Schupbach, K. Seiya, N. Tran, M. Schram, J. M. Duarte, Y. Huang, and R. Keller, Phys. Rev. Accel. Beams **24**, 104601 (2021).
244. P. J. Rousseeuw, J. Comput. Appl. Math. **20**, 53 (1987).
245. F. Irshad, C. Eberle, F. M. Foerster, K. v. Grafenstein, F. Haberstroh, E. Travac, N. Weisse, S. Karsch, and A. Döpp, arXiv:2303.15825 (2023).
246. B. Schölkopf, A. Smola, and K.-R. Müller, in *International Conference on Artificial Neural Networks* (Springer, 1997), p. 583.
247. E. J. Candès, X. Li, Y. Ma, and J. Wright, J. ACM **58**, 11 (2011).
248. A. Hyvärinen and E. Oja, Neural Netw. **13**, 411 (2000).
249. R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE, 2022), p. 10684.
250. R.-R. Griffiths and J. M. Hernández-Lobato, Chem. Sci. **11**, 577 (2020).

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press

251. A. Willmann, P. Stiller, A. Debus, A. Irman, R. Pausch, Y.-Y. Chang, M. Bussmann, and N. Hoffmann, arXiv:2106.00317 (2021).
252. P. Stiller, V. Makdani, F. Pöschel, R. Pausch, A. Debus, M. Bussmann, and N. Hoffmann, arXiv:2211.04770 (2022).
253. W. Rawat and Z. Wang, Neural Comput. **29**, 2352 (2017).
254. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, arXiv:1409.4842 (2015).
255. D. Balduzzi, M. Frean, L. Leary, J. P. Lewis, K. W.-D. Ma, and B. McWilliams, arXiv:1702.08591 (2018).
256. K. He, X. Zhang, S. Ren, and J. Sun, in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2016), p. 770.
257. C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, Proc. AAAI Conf. Artif. Intell. **31**, 4278 (2017).
258. A. Krizhevsky, I. Sutskever, and G. E. Hinton, Commun. ACM **60**, 84 (2017).
259. P. Viola and M. Jones, in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (IEEE, 2001), paper I–I.
260. L. T. Nguyen-Meidine, E. Granger, M. Kiran, and L.-A. Blais-Morin, in *Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)* (IEEE, 2017), p. 1.
261. S. Ren, K. He, R. Girshick, and J. Sun, arXiv:1506.01497 (2016).
262. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2016), p. 779.
263. L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, IEEE Trans. Pattern Anal. Mach. Intell. **40**, 834 (2018).
264. J. Long, E. Shelhamer, and T. Darrell, IEEE Trans. Pattern Anal. Mach. Intell. **39**, 640 (2016).
265. A. M. Hafiz and G. M. Bhat, Int. J. Multimedia Inform. Retr. **9**, 171 (2020).
266. K. He, G. Gkioxari, P. Dollár, and R. Girshick, IEEE Trans. Pattern Anal. Mach. Intell. **42**, 386 (2018).
267. R. Girshick, J. Donahue, T. Darrell, and J. Malik, IEEE Trans. Pattern Anal. Mach. Intell. **38**, 142 (2015).
268. C. Amorin, L. M. Kegelmeyer, and W. P. Kegelmeyer, Stat. Anal. Data Mining **12**, 505 (2019).
269. T. Pascu, in *LPA Online Workshop on Control Systems and Machine Learning* (2022).
270. X. Chu, H. Zhang, Z. Tian, Q. Zhang, F. Wang, J. Chen, and Y. Geng, High Power Laser Sci. Eng. **7**, e66 (2019).
271. I. Ben Soltane, G. Hallo, C. Lacombe, L. Lamaignère, N. Bonod, and J. Néauport, J. Opt. Soc. Am. A **39**, 1881 (2022).
272. Z. Li, L. Han, X. Ouyang, P. Zhang, Y. Guo, D. Liu, and J. Zhu, Opt. Express **28**, 10165 (2020).
273. J. Lin, F. Haberstroh, S. Karsch, and A. Döpp, High Power Laser Sci. Eng. **11**, e7 (2023).
274. E. Y. Sidky, L. Yu, X. Pan, Y. Zou, and M. Vannier, J. Appl. Phys. **97**, 124701 (2005).
275. H. Li, G. Liang, and Y. Huang, Comput. Phys. Commun. **259**, 107644 (2021).
276. S. Garca, S. Ramrez-Gallego, J. Luengo, J. M. Bentez, and F. Herrera, Big Data Anal. **1**, 9 (2016).
277. C. Shorten and T. M. Khoshgoftaar, J. Big Data **6**, 60 (2019).
278. E. Hüllermeier and W. Waegeman, Mach. Learn. **110**, 457 (2021).
279. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, J. Mach. Learn. Res. **12**, 2825 (2011).
280. 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, arXiv:1603.04467 (2016).
281. 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, Adv. Neural Inform. Process. Syst. **32**, 8024 (2019).
282. J. Herzen, F. Lässig, S. G. Piazzetta, T. Neuer, L. Tafti, G. Raille, T. V. Pottelbergh, M. Pasieka, A. Skrodzki, N. Huguenin, M. Dumonal, J. Kościsz, D. Bader, F. Gusset, M. Benheddi, C. Williamson, M. Kosinski, M. Petrik, and G. Grosch, J. Mach. Learn. Res. **23**, 1 (2022).
283. P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors , Nat. Methods **17**, 261 (2020).
284. J. Blank and K. D. pymoo, IEEE Access **8**, 89497 (2020).
285. F. Biscani and D. Izzo, J. Open Source Software **5**, 2338 (2020).
286. M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, Adv. Neural Inform. Process. Syst. **33**, 21524 (2020).
287. T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (ACM, 2019), p. 2623.
288. T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, arXiv:2005.14165 (2020).

https://doi.org/10.1017/hpl.2023.47 Published online by Cambridge University Press