4 Machine Learning as a Solver for DMFT

Cedric Weber
King's College London
The Strand, WC2R 2LS London

Contents

E. Pavarini, E. Koch, A. Lichtenstein, and D. Vollhardt (eds.)
Dynamical Mean-Field Theory of Correlated Electrons
Modeling and Simulation Vol. 12
Forschungszentrum Jülich, 2022, ISBN 978-3-95806-619-9
http://www.cond-mat.de/events/correl22

1 Introduction

Widespread interest has been devoted in the last three decades to strongly correlated materials, which are being used in emerging technologies, such as spintronics, quantum computing and high temperature superconductivity for example. They are characterized by strong electronic interactions between their d and f-band valence electrons. The interplay between charge localization (Mott physics) and itinerant behavior (quasi-particles) provides a challenge for standard electronic theories, such as density function theories. In recent years significant progress in understanding the underlying physics of strong electron correlation effects has been made by the dynamical mean-field theory (DMFT) method [1], in particular with the harnessing of DMFT to widespread materials modelling methods, such as density functional theory (DFT), with the DFT+DMFT combined approach [2].

Despite this formidable achievement, practical challenges remain with the DFT+DMFT approach. At the heart of the theory is the choice of the quantum engine that solves the many-body Anderson Impurity Model (AIM), an underlying model that provides the local Green function of the compound of interest via a self-consistent mapping. Currently, methods of reference at finite temperature are based on statistical sampling, with for instance the continuous-time quantum Monte Carlo solver [3]. The latter provide exact results (within sampling error bars) on the imaginary axis for the fully rotationally-invariant AIM Hamiltonian.

Other solvers can provide robust information on the real axis, such as numerical renormalization group approach [4], but their application remains currently challenging for the case multi-orbital systems for the DFT+DMFT approach.

Another well-known method that provides information on the real-axis is the exact diagonalization (ED) approach. Here, the DMFT hybridization is approximated by a finite number (N_b) of effective bath orbitals. In practice, N_b is restricted because of the exponential growth of the Fock space with the total number of sites (bath and impurity). The growth of the Hilbert space is mitigated by Lanczos-based algorithms, which utilize the sparse nature of the Hamiltonian, and to a large degree enable routine calculations for systems up to $N_s \approx 12$ [5]. Overall, the limitation N_s limits the scope of applicability of this method, and typically discretization effects remain large for multi-orbital systems.

In summary, the quantum engine at the heart of DMFT remains an active topic of research, and there is no single silver bullet that provides a general solution of the AIM problem for all contexts and all parameters ranges. Generally, with a few exceptions, most quantum solvers are also computationally demanding, adding a significant overhead to the DFT calculation. Until the latter is addressed, high-throughput material design will remain beyond the reach of most researchers, without access to dedicated large HPC infrastructures: the quantum solver used to provide the solution of the embedding theory, as for every calculation applied to each material of interest, has to be solved hundred's of time with a large inherent computational cost. Furthermore, if approaches such as DMFT are made widely available to engineers and scientists, it would be repeated all across the world, at the cost of thousands of computing hours for every single calculation, generating large hallmarks in terms of energy consumption, data storage and

computational slow-downs. Approximate methods provide cost effective alternatives, such as perturbation based theories, but they are qualitatively wrong in known limits of the theory.
The fundamental problem lies in the fact that the effects of the Coulomb interaction and of the specific material cannot be separated. Otherwise, one could calculate the interaction contributions once and forever, store them and add them every time a new material is calculated.
The most prominent example of the strategy is indeed the local density approximation (LDA) to density functional theory (DFT), where the Kohn-Sham exchange-correlation potential is taken from a homogeneous electron gas that is calculated at the density of the real system in the same point. In this way, DFT profits from the existence of tabulated and interpolated Quantum Monte Carlo (QMC) results, shared with the entire scientific community.
In the latter context, the Anderson impurity model (AIM) plays a central role in the dynamical mean-field theory (DMFT), very similarly to the role of the local density approximation in DFT, where the exchange functional involves solving the quantum problem of a gas of interacting electrons. Contrary to DMFT however, where the AIM is solved every single time without data storing or sharing, the success of DFT has been established by a simple but efficient strategy: the difficult problem of the interacting electronic gas has been solved at a large computational cost with Quantum Monte Carlo, once and for all, and for everyone, shared with the wider scientific community for the benefit of all [6].
It is very surprising that only moderate work has been done on storing and sharing solutions of the AIM obtained at large computational costs (typically in the several thousands of core-hours for every single-point calculation on a given compound), as the obtained many-body solution of the AIM is not specific to a given material. Furthermore, repeatedly solving the AIM correlated problem at large computational cost is also required when only small variations are introduced by finite displacements (for lattice dynamics properties). This limits the scope of predictions for structure optimization or other interesting applications under applied constraints, such as external pressure, due to the overwhelming number of required calculations, when relaxing different chemical compositions with numerous single-point calculations of the same chemical composition are required. This latter fundamental limitation of DMFT, and more generally quantum embedding approaches, can be addressed with advances in the field of data science.
Application of machine learning (ML) to quantum physics as neural networks and gaussian processes to accelerate material discovery have been well studied, for instance for accelerating DFT [7–9] optimizing QMC in terms of speed [10–12] and accuracy [13], and similarly for analytic continuation [14].
In the context of DMFT, the pioneering work of Arseneault *et al.* [15] has opened new avenues in applying ML to the Anderson impurity model for a broad range of parameters. Recently, the application of neural networks (NN) combined with different exact solvers has further established the validity of ML approaches in the context of DMFT [16–18]. These approaches have pioneered in applying ML to quantum many-body systems.
We note that ML and its applicability is limited in terms of offering an overall general solution of quantum many-body systems, for reasonably sized data-bases, due to the rich and complex nature of the many-body problem.

Here, we review the literature on machine learning and neural networks within the context of condensed matter. We also address a simpler task than providing the whole solution of the AIM from a data-science perspective. Instead, we use here physically inspired approximate and cost efficient solvers to DMFT, and outline an approach to provide many-body corrections to account for the obtained errors of the fast solver [19]. The credit to this work goes to the lead author of this work, Dr Evan Sheridan (Phasecraft), and to Dr Francois Jamet (UK National Physics Laboratory) and Zelong Zhao (King's College London).

1.1 Supervised learning and linear regression

We illustrate here the concept of supervised learning within the framework of linear regression. Let's consider for instance a data set that we'd like to fit with a simple regression model. For sake of illustration we'll consider data obtained from electric vehicles, where we relate the mileage in miles per gallon equivalent (MPGe) to the vehicle weight and battery capacity [20].

Vehicle List		
Vehicle weight (Kg)	Battery Capacity (kWh)	Mileage (MPGe)
1000	54	108
1500	81	103
2000	108	98
2500	135	93
3000	162	88
3500	189	83
4000	217	78

The data set is composed of two-dimensional vectors, where x_1^i is the vehicle weight, and x_2^i the battery capacity of a given vehicle. We note that the given features need to be chosen with care for a given problem. To perform any sort of learning, we need to represent a model function for the mileage

$$h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \,. \tag{1}$$

The θ parameters, or weights, are the model parameters that will be learned throughout the supervised learning process. Introducing the extra term $x_0 = 1$, the notation simplifies to

$$h_\theta(\mathbf{x}) = \sum_{i=0}^{d} \theta_i x_i \,. \tag{2}$$

In the spirit of regression, we need to identify the optimal parameters θ_i that provide the best model for the known dataset (training set), and ultimately for inferring the mileage on future vehicles (inference process). How can we learn from the available data at hand, and obtain the model parameters? To address this question, we need to define a figure of merit of our model, or a cost function

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2, \tag{3}$$

where y^i are the known mileage obtained from the table above. Minimizing the cost function will hence provide the theoretical model. This can be achieved with a gradient descent algorithm for instance, starting from a initial guess for θ, and repeatedly updating the parameter values by following the steepest gradient,

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta). \tag{4}$$

The gradient descent update is performed over all parameters θ_j simultaneously and the procedure iterated until convergence is achieved. The parameter α plays the role of a *learning rate*, indeed for small α changes in the model parameters θ are small and progress is slow, with the caveat that a large number of iterations is required to reach convergence, whereas for large α changes in the model parameters are rapid, but convergence might be hampered from sudden jumps in the iteration process. To implement the algorithm, one can analytically calculate the derivative and one can easily check that the following is obtained

$$\frac{\partial}{\partial \theta_j} J(\theta) = \big(h_\theta(x) - y\big) x_j, \tag{5}$$

and combined with equation (4) we obtain the *training rule*, also known as Widrow-Hoff rule

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}. \tag{6}$$

As mentioned above, the learning rate is proportional to the proportional coefficient α, but perhaps counter-intuitively the amplitude of the learning is proportional to the error rate $y^{(i)} - h_\theta(x^{(i)})$. Thus, this algorithm learns most from large deviation from the sample, i.e., when our prediction has a large error and $h_\theta(x^{(i)})$ deviates most from $y^{(i)}$. Some practical considerations are absent from the discussion above. In particular, the parameter θ can be updated for every measurement or known data point of the training set, albeit the latter is in reality finite. This highlights the importance of the learning rate. Furthermore, our model is limiting in terms of dependencies and extensions: what if we would like to add a parameter in the model related to the weight of the battery, which is a function of both the total vehicle weight and battery capacity? Such a parameter should feed from the previous variable and provide a higher level model parameter. In the next section we will extend this simple model and learning process to a more general framework that allows for this flexibility. The formulae obtained above are of course nothing else than the well known linear regression method, but it provides a means to set the terminology and general extension to neural networks, discussed in the section hereafter.

1.2 A single layer neural network: the perceptron model

Neural networks are a flexible ensemble of data-driven models, largely inspired by the human's brain network of synapses and neurons, that provide non-linear neural connections. In contrast to a biological neuron, inside the artificial neural network the neuron, or perceptron, is in the form of a simple function whose operation is to take an input vector $\mathbf{x} = \{x_1, \ldots, x_n\}$ (the *feature vector*) and activate a logical threshold if the signal is large enough,

$$f(\mathbf{x}) = \sigma\big(\mathbf{x}.\mathbf{w} + b\big), \tag{7}$$

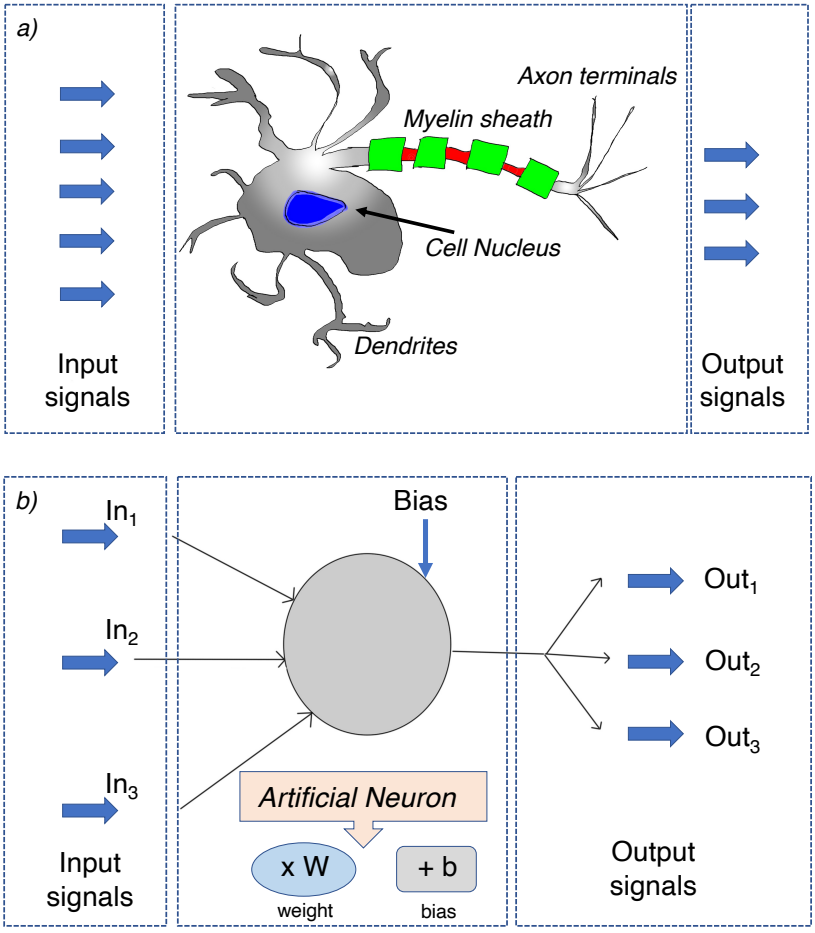

Fig. 1: *a) Schematic of a biological neuron, which consist of a cell nucleus, synapses connected via dendrites to the cell, myelin sheath that embeds the axon that ultimately provides the output signal via the axon terminal to another neuron. b) Simplified artificial neuron model: input signals In_i are multiplied by a proportional weighting factor W and a constant bias b is added to the signal, providing the output Out_i that connect with the other neurons.*

where σ is a continuous activation function, $\mathbf{w}$ is a set of parameters that are specific to the neuron, and b is the bias parameter. Connecting a network of neurons together and adjusting the weights to match the value of a target output provide a mean to use the network to build non-linear predictive responses for different given inputs, which is foundation of learning.

The perceptron model was originally introduced by Frank Rosenblatt, who simulated and built purpose hardware for this model in the early 1960s that provided a direct and parallel implementation of perceptron learning [21]. This model is the first neural network learning model introduced, which is simple and limited, but provides the basic concepts and is a good learning tool. The original motivation for deriving this theory was related to the physiology of the brain learning process, and in particular pattern recognition. The theory is based on a simplified model for the brain neuron: the latter consist of a complex interplay between input signals carried by synapses, interconnected with the neuron cell which provides a time-dependent output signal transported via the axon terminals (see Fig. 1.a). Many complex physiologic phenomena occur via the brain neuron cells in the learning process. The simplest model of an artificial neuron con-

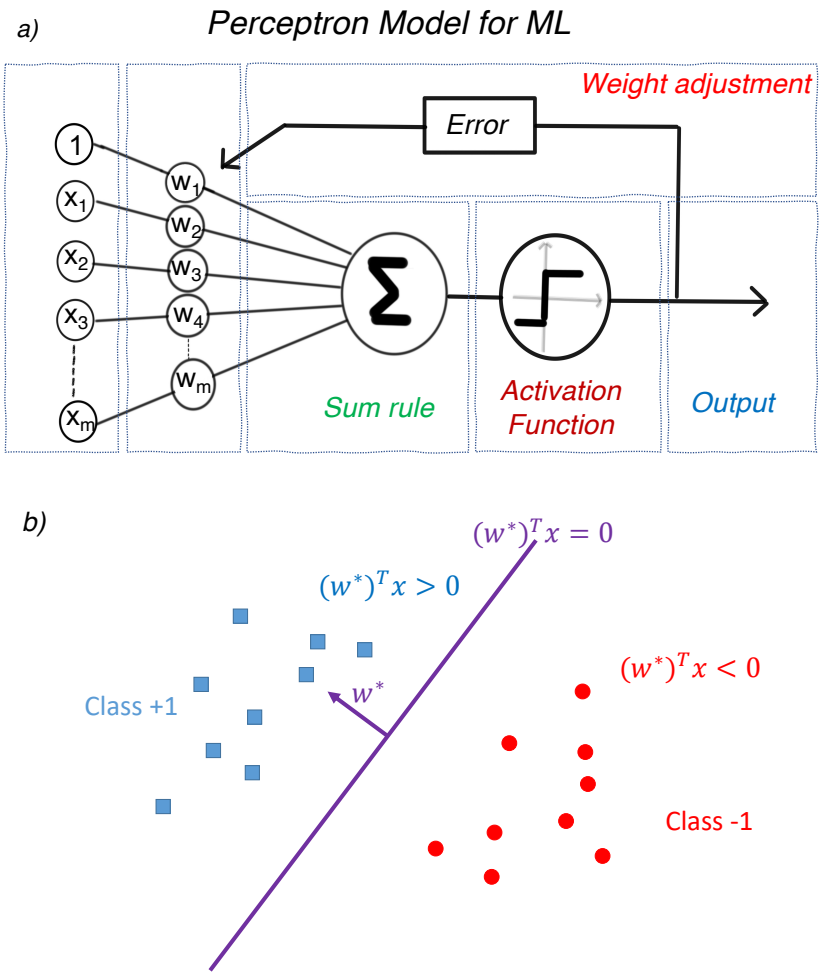

Fig. 2: *a) Schematic of the perceptron model. Inputs x_i are weighted with w_i, and collected by a sum-rule Σ, before entering a threshold logic unit and triggering the activation function. The output is then obtained and compared to the training set. The perceptron learns by the error correction method, where the weights are updated based on the obtained error. b) A typical classification task where the training set consists in a set of data points labeled as circles or squares. After the learning process, the perceptron weights w_i correspond to the equation of the separation line.*

siders a simple proportional relation between input and output signals via a weight coefficient w_i and a constant applied bias b_i (see Fig. 1.b). This model omits the time-dependence of the output signal and many other factors, but provides a basic building block for inter-neuron connections. Typically, the perceptron model consist of a layer of artificial neuron cells, connected to a set of input signals x_i (see Fig. 2a). To mimic the learning process, a summation is applied to the neuron layer, which collects the weighted sum of all input signals. A threshold logic unit is then applied which determines the outcome of the final output binary signal, typically the output signal being $z = 1$ if the learning outcome is positive, and $z = 0$ in the alternative. This provides typically a means to classify data in two categories (classifier). A typical example is a set of data points in Euclidean space which are delimited in two classes, as to whether they

lie above a delimiting line, or below it (see Fig. 2b). The line coefficients w_i are unknown, but instead we know for a group of points whether they belong to the class $+1$ or -1.
In this example, we are provided with a given training data set $\big\{(x_i,\, y_i)\,\big|\, i{=}1,\ldots,n\big\}$. We define the activation function $f_w(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

- $y = +1, \qquad$ if $\mathbf{w}^T\mathbf{x} > 0$
- $y = -1, \qquad$ if $\mathbf{w}^T\mathbf{x} < 0$

The prediction of the perceptron model is the sign of the activation function $\text{sign}\big(f_w(\mathbf{x})\big)$. The aim of this approach is to learn from the data set and minimize the classification error.
We will use a two neuron model, and following the general recipe, we will set the weights to random initial values: $w_1 = 0.4$ and $w_2 = -0.2$ (note that weights can be either positive or negative). Our training set is set as follow:

Training set		
x_1	x_2	outcome
0.2	0.3	1
0.4	0.1	0

The learning process occurs by testing the algorithm on the training set and to adjust in turns the network weights in the learning process. Weights are typically adjusted by comparing the prediction of the network on a given data point, and correcting for errors obtained in the evaluation. We provide here a simple example and recipe to optimize weights in the single layer neural network, with a simple learning algorithm and objective function. To be more specific, we use the objective function $\mathcal{C}$

$$z \;= 1, \qquad \text{if} \qquad \sum_{i=1}^{n} x_i\, w_i \leq \theta$$

$$z \;= 0, \qquad \text{if} \qquad \sum_{i=1}^{n} x_i\, w_i > \theta$$

where θ is the threshold value. This part defines the logical activation function that converts the signal, modulated by the network weights, into a prediction. The task of the learning process is to train the network weights, for a given objective function, such that the training set is reproduced accurately. For the sake of illustration, we use here a threshold value $\theta = 0.1$.

Training set					
x_1	x_2	ω_1	ω_2	Prediction $\mathcal{P}$	Dataset $\mathcal{D}$
0.2	0.3	0.4	–0.2	1	1
0.4	0.1	0.4	–0.2	1	0

Applying our randomized neural network, we observe that the first training data point is actually well classified by the network with original choice of weights. However, for the second data point, our network produces a wrong prediction. After every error of the network, we perform a weight update with the following learning rule

$$\Delta w_i = \alpha\,(t - z)\,x_i, \tag{8}$$

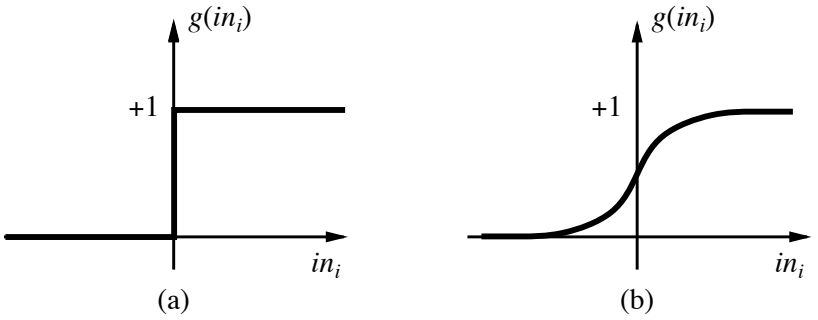

Fig. 3: *a) Activation step-function. b) Activation sigmoid function.*

where t is the target value (training set), z the current output, α the learning rate, w_i the weight associated with input i, x_i the corresponding input value. We note the direct correspondence with the parameter update obtained in equation (6) in the steepest descent approach. The following pseudo-code provides the general approach for a single or multiple layer neural network

Algorithm 1 Neural network pseudo-code

inputs for sample data point: θ,w_i,x_i
Require: $\mathcal{C} = 0$
call Evaluation function: x_i, α, θ, w_i, $\mathcal{D}_i$ output w_i
Prediction $\mathcal{P}_i \leftarrow \sum_{i=1}^{n} x_i w_i \leq \theta$
Cost function $\leftarrow \|\mathcal{D}-\mathcal{P}\|$
Learning $w_i \leftarrow \alpha\,(\mathcal{D}-\mathcal{P})\,x_i$
Iterate over training set

The weights obtained from the neural network will eventually provide a means to predict the class of an unknown data point y_i. The weights have a very simple geometrical interpretation, they represent the line parameters (see Fig. 2.b). In the simple perceptron model, the relation between outputs and inputs remains linear, due to the limited complexity of the model.
We note that in our approach, the choice of the threshold value θ and logical rule to determine whether the weighted signal falls within class A or B is entirely arbitrary. The mathematical formulation of the logical threshold unit is denoted as *activation function*. In general, there is a breadth of possible choices available and studied in the literature, providing different learning efficiency and resilience towards noise, typical examples are the step function and the sigmoid function $(1/\,(1+e^{-z}))$, ReLU $(\max(z,0))$, or tanh (see Fig. 3).
To expand the scope of this approach, and allow for identifying non-linear boundaries between more complex sample sets, the perceptron model can simply be extended by allowing for several neuron layers between inputs and the logical threshold function. The current is simply modulated several times, by the weights of the respective layers. Furthermore, we can also allow for connections between a neuron of one layer with multiple neurons of the next layer, allowing for a large number of weight parameters. This is the realization of a so-called *neural network*.

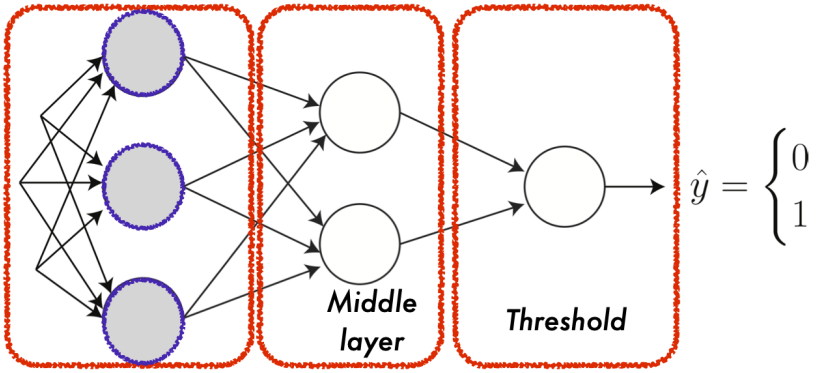

Fig. 4: *Simple extension of the perceptron model, where we have now an intermediate neuron layer between the input fully connected layer and the binary output.*

1.3 Neural networks

A neural network is a direct extension of the perceptron model. It has two main components: (i) the network architecture in terms of number of layers, neurons per layer, and how neurons are inter-connected, and (ii) the parameters defining connections (weights), with the task to learn the parameters for achieving a given task. In our application to DMFT, this task will consist in learning the errors of a given approximate solver to the Anderson impurity model.

In our context, the input will consist of a Green function $G(\tau)$ represented in imaginary time, which is essentially a vector of d dimension $\left(x_1^{(i)}, x_2^{(i)}, ..., x_d^{(i)}\right)$. We'll come back to our main aim in the next sections, but for now we extend the example of the perceptron model where we want to classify an input in a binary class $\hat{y} = 0$ or $\hat{y} = 1$ (see Fig. 4). On the left of this figure, the input is connected to the first layer, the *fully connected layer*. The second layer is denoted as a *hidden layer*, as its presence merely provide additional degrees of freedoms to propagate the information forward to the logical threshold unit. It is worth noting that we have so far only considered forward propagation of the information throughout the network. After doing a first single forward pass through the network, for a given initial choice of weights w_i and a given input vector x_i, we need to update the parameters for the learning process.

Here, we need to generalize the learning formula introduced in the context of the perceptron model. This extension leads to the concepts of training loss, and validation loss. The former is a metric used to assess how the model fits the training data, i.e., it assesses the error of the model on the training set. It is worth noting that the training set is a portion of the entire dataset used to initially train the model. Computationally, the training loss is calculated by taking the sum of errors for each example in the training set. It is also important to note that the training loss is measured after each batch, that is usually visualized by plotting a curve of the training loss after each update of the weights. The latter (validation loss) is a metric used to assess the performance of the learning model on the validation set. The validation set is another portion of the dataset set aside to validate the performance of the model, usually a smaller portion of the dataset as the largest chunk is used to train the model instead (typical splits of the entire dataset in validation/training are 20% validation and 80% training).

The loss function used in neural networks is based on the binary cross-entropy formula

$$\mathcal{L} = -\big(y_i \log_2(\hat{y}_i) + (1-y_i)\log_2(1-\hat{y}_i)\big), \tag{9}$$

where y represents the expected outcome and $\hat{y}$ the outcome produced by the model. Let's have a look at a simple example: for a neural network that tries to determine whether a picture contains a cat, the outcome is either of 1 (cat) or 0 (no cat). With a sample that has two pictures, the first of which contain cats, whilst the second doesn't. Let's imagine that the neural network is 80% confident that the first image contains a cat: $y = 1$ and $\hat{y} = 0.8$. The loss function in equation (9) gives $\mathcal{L} = 0.32$. For the second image, the network gives a 90% probability that there aren't any cats in the picture, $y = 0$ and $\hat{y} = 0.1$, with $\mathcal{L} = 0.15$. The loss function is designed such that either the first term $y_i \log_2(\hat{y}_i)$ or the second term $(1-y_i)\log_2(1-\hat{y}_i)$ are naught or small when the network has a large confidence in asserting the classification, whilst the loss function is large in uncertain situations. The loss function can be averaged over the training sample (or the validation sample), leading to the overall cost function $\mathcal{C}$

$$\mathcal{C} = -\frac{1}{N}\sum_{i=1}^{N}\big(y_i \log_2(\hat{y}_i) + (1-y_i)\log_2(1-\hat{y}_i)\big). \tag{10}$$

For DMFT we are however not focusing on binary classification, and instead our predicted and model values are in general real. The fairly straightforward extension to a real number can be achieved simply by generalizing the cost function with a regression model, for instance

$$\mathcal{C} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2. \tag{11}$$

1.4 Back-propagation

Once the error is established, the weight update is less obvious than in the case of the perceptron model, where the error obtained on each components x_i could directly be linked and associated with a well defined weight w_i.

Although the gradients of the loss function will provide eventually the weight update, as a generalization of the linear update that we have seen in the previous section, the connection between error and weights is convoluted due to the multiple intermediate layers. Let us first introduce more specific notation for the neural network. Fig. 5 provides a schematic of a maximally connected feed-forward network, where the web of neuron connections is illustrated, with the associated activation values h and the prediction made in the final layer h_1^4. In what follows, we consider training the network on input data $\mathbf{X} = \{x_1, \ldots, x_N\}$ and their associated outputs $\mathbf{Y} = \{y_1, \ldots, y_N\}$. The first-pass through the network consists of assigning $h_i^0 = x_i$ and evaluating the activation functions at each layer of the network as

$$h_j^l = \sigma\Big(\sum_k w_{jk}^l h_k^{l-1} + b_j^l\Big) = \sigma\big(z_j^l\big)\,, \tag{12}$$

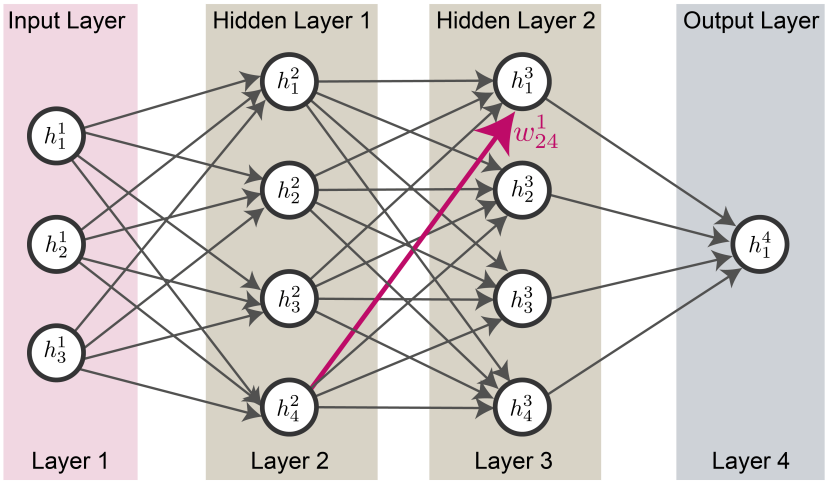

Fig. 5: *Schematic of a feed-forward neural network. An input vector is propagated through the system, via a number of hidden layers, triggering the activation functions h_i^j of each neuron i in layer j.*

where $z_j^l = \sum_k w_{jk}^l h_k^{l-1} + b_j^l$ which can be more compactly expressed as,

$$\mathbf{h}^l = \sigma\big(\mathbf{w}^l \mathbf{h}^{l-1} + \mathbf{b}^l\big). \tag{13}$$

At the end of the first-pass through the network the aforementioned cost function is evaluated,

$$C = \frac{1}{2N} \sum_i^N \big|y_i - h_i^L\big|^2, \tag{14}$$

where h_i^L is a nested function of x_i with many intermediate evaluations of σ,

$$h_i^L = \sigma^L\Bigg(\sum_k w_{ik}^L \sigma^{L-1}\Big(\sum_k w_{ik}^L \sigma^{L-2}\big(\dots x_i \dots\big) + b_i^{L-1}\Big) + b_i^L\Bigg). \tag{15}$$

Given the nested structure of this function and the sheer number of parameters that $\mathbf{W}_{ij}$ can have, it is prohibitive to find analytical solutions for the combination of weights that minimize equation (14). Instead, established gradient descent methods are applied [22] and define what is now known as the *backpropagation* approach.

The central question of the backpropagation method is to calculate the variation of the cost function with respect to all of the network parameters, $\partial C/\partial w_{jk}^l$ and $\partial C/\partial b_j^l$, and to use these gradients to update the weights. The first step to obtaining these expressions is to express the error in the j-th neuron of layer l as

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \frac{\partial C}{\partial h_j^l}\frac{\partial h_j^l}{\partial z_j^l} = \frac{\partial C}{\partial h_j^l}\,\sigma'\big(z_j^l\big), \tag{16}$$

where we have applied the chain rule. In this form, δ_j^l doesn't exploit the connectivity of the overall network. Indeed, calculating

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{h_j^l}\frac{\partial h_j^l}{\partial w_{jk}^l} = \frac{\partial C}{\partial h_j^l}\,\sigma'\big(z_j^l\big)\,h_k^{l-1} = \delta_j^l\,h_k^{l-1}, \tag{17}$$

shows that variation of the cost in layer l with respect to its weight is dependent on the activated output in the preceding layer. Hence, relating the errors from layer-to-layer can allow for a systematic way to calculate the variation of the cost in each layer of the network

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1}\, w_{kj}^{l+1}\, \sigma'\big(z_j^l\big). \tag{18}$$

Similarly, $\partial C/\partial b_j^l$ can be found as $\frac{\partial C}{\partial b_j^l} = \delta_j^l$. All weights in the network can then be updated by gradient descent in the following manner,

$$w_{jk}^l \to w_{jk}^l - \eta\, \frac{\partial C}{\partial w_{jk}^l}, \quad \text{and} \qquad b_j^l \to b_j^l - \eta\, \frac{\partial C}{\partial b_j^l}, \tag{19}$$

where η is the so-called *learning rate*. The origin of the name *backpropagation* refers to the equations above as for the update δ_j^l one must first know all errors in the subsequent layer δ_k^{l+1}, and hence the error propagates backward through the network.
In summary, the learning process in the neural network consists of repeated forward- and backward-passes throughout the network, after each pass the cost function is reduced. The forward-pass propagates the input forward for the evaluation of the cost function, while the backward-pass updates the network weights starting in the output layer and back-propagating the information to the input layer, that in turn implements the weight updates that reduce the cost function on the next forward pass.

2 Generating a quantum database for the Anderson impurity model

We have now introduced all the concepts in the field of data science required for designing a data-driven approach for solving quantum many-body hamiltonians. To apply the learning process outlined above, we need first to decide on a compact representation of the many-body quantities that will be used in the neural network.

2.1 Polynomial basis method

One way to represent the Green function in a compact formulation is via a polynomial support basis. We expand $G(\tau)$ in an arbitrary orthogonal polynomial basis $P_i^{(k)}\big(x(\tau)\big)$ (e.g., Legendre, Chebyshev, or else) where i is the polynomial order, k is the species and $x(\tau) = 2\tau/\beta - 1$ is a transformation to absorb the temperature dependence from $[-1, +1]$ to $[0, \beta]$. The expansion is

$$G^{(k)}(\tau) = \sum_{i \geq 0} P_i^{(k)}\big(x(\tau)\big)\, G_i^{(k)}. \tag{20}$$

Applying the orthogonality constraints obeyed by the polynomials

$$\int_0^\beta P_i^{(k)}\big(x(\tau)\big) P_{i'}^{(k)}\big(x(\tau)\big) W\big(x(\tau)\big)\, d\tau = \tilde{W}(i)\, \delta_{i,i'}, \tag{21}$$

provides a means to obtain the basis coefficients that can be calculated as

$$\int_0^\beta G^{(k)}(\tau)P_i^{(k)}\big(x(\tau)\big)W(x(\tau))d\tau = \int_0^\beta \sum_{i'\geq 0} G_{i'}^{(k)}P_{i'}^{(k)}\big(x(\tau)\big)P_i^{(k)}\big(x(\tau)\big)\,W\big(x(\tau)\big)d\tau = \tilde{W}(i)G_i^{(k)}. \tag{22}$$

For the purpose of this discussion one restrict the analysis only to the Legendre polynomials, where $W(x) = 1$ and $\tilde{W}(i) = 4/\beta/(2i{+}1)$.
With the G_i, we can express $G(\tau)$ on an arbitrarily fine imaginary time τ grid, absent from discretization constraints. For a given $G(\tau)$, the procedure outlined above provides us with the g_l coefficients, or as an alternative a simple fitting procedure can be achieved, which amounts to the minimization of the function

$$\min_{\{g_l\}}\big(G(\tau) - G_{\text{FIT}}^{\{g_l\}}(\tau)\big), \tag{23}$$

where the fitted (model) function $G_{\text{FIT}}^{\{g_l\}}(\tau)$ is parametrized by the basis coefficients g_l, i.e.

$$G_{\text{FIT}}^{\{g_l\}}(\tau) = \sum_{l\geq 0}^{N_l} \frac{\sqrt{2l{+}1}}{\beta} P_l\big(x(\tau)\big)g_l. \tag{24}$$

It is straightforward to find $G_{\text{FIT}}^{\{G_l\}}(\tau)$ using the conjugate gradient method for example, as the number of Legendre coefficients N_l is generally of the order of a few tenths, typically $N_l \approx 20$. This procedure is indeed exceptionally computationally efficient. Furthermore, it can also be used to filter out noise if a Monte Carlo solver is used to generate the database, shifting the paradigm of dealing with a statistical problem to that of an optimization one. It allows the advantage of including *a-priori* information, such as tail exponents etc.

2.2 Generating the validation dataset with exact diagonalization

Now that we have introduced the representations of our dataset, we need to provide a training set of reference data against which the neural network can learn and validate. As a reference solver for the AIM, it is essential that the solver can efficiently compute solutions in fast and stable ways over a wide range of parameters (Coulomb U, bandwidth W and inverse temperature $\beta = 1/T$). A variety of solvers are available for use in DMFT calculations, and they can be selected to match the computational resources available to complete a calculation in a time-efficient way. As we will see hereafter, we will consider two families of solvers

i. A selection of fast approximate solvers that have low computational cost an provide a reasonable solution of the AIM in some limits of the parameter space, failing in other regions of parameters (e.g. strong or weak interaction limits).

ii. A choice of benchmark solver against which the solvers in (i) above can be compared.

We discuss first the choice of an exact solver for validation and training in (ii). We use here the exact diagonalization method, a typical approach known for solving the AIM in DMFT [23],

where the infinite lattice surrounding the impurity site is approximated by a discretized bath of finite size N_b. The first step is to parametrize the Weiss field $\mathcal{G}_0$ for each orbital and spin in terms of a finite number of bath orbitals by approximating the Weiss field in terms of a non-interacting Anderson impurity model

$$\mathcal{G}^{-1}_{\text{And},m}(i\omega_n) = i\omega_n + \mu - \epsilon^{\text{imp}}_m - \sum_{k=1}^{k=N_b} \frac{V_{m,k}V^*_{m,k}}{i\omega_n - \epsilon_{m,k}} \tag{25}$$

where k is the index for the bath level, m is the index for each orbital/spin. This entails minimizing a distance between the Weiss field and the parametrized impurity Green function obtained in equation (25), using a cost function

$$\chi^2(\epsilon_k, V_k) = \sum_{n=0}^{n=n_c} \mathcal{A}_n \left| \mathcal{G}_{\text{And}}(i\omega_n;\, \{\epsilon_k, V_k\}) - \mathcal{G}_0(i\omega_n) \right|^2. \tag{26}$$

For using ED as a solver for DMFT, it is common practice to weight the cost function towards smaller imaginary frequencies by using a prefactor $\mathcal{A}_n \approx 1/i\omega_n^2$. This avoids fitting cost on the asymptotic regions of the Green function, which are known analytically, but instead provides a good solution in the low energy regime.

Once a set of ϵ_k, V_k has been identified, the calculation proceeds as a standard brute force matrix diagonalization by scanning quantum sectors of the AIM (either total spin S^z or numbers of up and down particles). The focus in DMFT-AI is however reversed: one can limit the database to the large ensemble of parameters ϵ_k, V_k which are tractable with typically 6, 7 or 8 bath sites, such that the solution of the AIM remains exact without the need for iterative solvers (Lanczos, Arnoldi, ...) and the computational cost reasonable. We hence limit ourselves to a large but finite set of corresponding hybridization functions that will be used to train the neural network.

We now turn to the discussion of the *fast and approximate solvers*. Perturbation theory is a well-known and successful diagrammatic method for solving quantum many-body problems in the weak-coupling limit. It is quite often the first port of call in a scientist's arsenal when tackling the quantum many-body problem. The goal of weakly perturbative methods for the AIM is to approximate $\Sigma(i\omega_n)$ analytically with diagrammatic expansions in the Coulomb repulsion U/t for (t is the hopping term in the Hubbard model)

$$G_0^{-1}(i\omega_n) = \Sigma(i\omega_n) + G^{-1}(i\omega_n). \tag{27}$$

Weak coupling expansions in U/t up to second order, $\mathcal{O}(2)$, were successfully used for the AIM [1] to capture the main features of the Mott transition at strong coupling in the nonperturbative regime. This only applies at half-filling, and can be attributed to the simple form of the atomic Green function in the $t/U \to 0$ limit being proportional to U^2 [1]. Nonetheless, iterative perturbation theory (IPT) is extremely useful for generating solutions for the AIM at low computational cost, in spite of the parameter regimes where the solution can be qualitatively wrong. Specifically, the second order perturbation of $\Sigma(i\omega_n)$ at inverse temperature β and half-filling is given by,

$$\Sigma^{\text{IPT}}(i\omega_n) = \Sigma^1(i\omega_n) + \Sigma^2(i\omega_n) = \frac{U}{2} + U^2 \int_0^\beta d\tau\, e^{i\omega_n\tau} G_0^2(\tau)\, G_0(-\tau), \tag{28}$$

where $\Sigma^{1,2}(i\omega_n)$ consist of the non-skeleton $\mathcal{O}(2)$ representations of the self-energy. Iterating over (27) with the above form of the self-energy is the foundation of the IPT theory, that successfully captures the Mott transition. Higher order diagrams can also be readily incorporated into this approach, to provide further corrections, but the complexity rapidly increases with the diagrammatic order, limiting the scope of manually correcting this approach. Practically, this amounts to replacing (28) with

$$\Sigma(i\omega_n) = \Sigma^1(i\omega_n) + \Sigma^2(i\omega_n) + \Sigma^3(i\omega_n), \tag{29}$$

where $\Sigma^3(i\omega_n)$ encapsulates all of irreducible third-order processes allowed, coming at an additional computation cost due to calculating the integrals associated with the higher-order diagrams and their additional interaction vertices. IPT is a good example to illustrate the data-driven approach outlined in these notes: the neural network that we will discuss below learns the error obtained in IPT and provides a highly non-linear solution in a multi-dimensional space to account for $\Sigma^n(i\omega_n)$ with $n > 2$, absent in standard IPT.

There are of course other approaches than neural networks that deal with corrections beyond second-order perturbation theories, and out of half-filling. For instance the Non-Crossing Approximation (NCA) is the lowest order strong-coupling perturbative method that sums up all diagrams without crossing hybridization lines. In this scheme the propagator of the impurity is mapped to an integro-differential Volterra equation that is solved for the strongly-coupled form of the self-energy in equation (27).

Both NCA, IPT, and the exact-diagonalization solver with a very small number of bath sites (typically zero, also known as Hubbard-1, or with $N_b = 1, 2, 3$ bath sites) represent a valid ensemble of approximate solvers which all introduce a negligible overhead in terms of calculations, and also all need corrections for providing quantitatively accurate solutions of the AIM, covering both the weak- and strong-coupling limits in the phase-space of the AIM. Finally, we note that those solvers are tractable and can also provide solutions in both real and Matsubara frequencies, but we'll limit the discussion in a first instance to the imaginary time formalism hereafter.

2.3 Data representation

The construction of a high-quality database of training samples is of key importance for any data-driven approach. Specifically, there must be sufficient *representative* samples, such that after the training process the inference process will produce the most likely outputs. Strategies for generating databases in machine learning are key for the success of any data-driven approach, and require great care in identifying robust and well thought strategies.

Before presenting the database construction at great lengths, we first need to view the AIM from a data-science perspective. Bearing this in mind, we look at the AIM from a black box perspective, and regard it only in terms of its inputs and outputs. In doing so, the AIM merely provides a relation between input $\mathbf{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_{N_S}}\}$ and output samples $\mathbf{Y} = \{\mathbf{y_1}, \ldots, \mathbf{y_{N_S}}\}$, where N_S is the number of database samples or images.

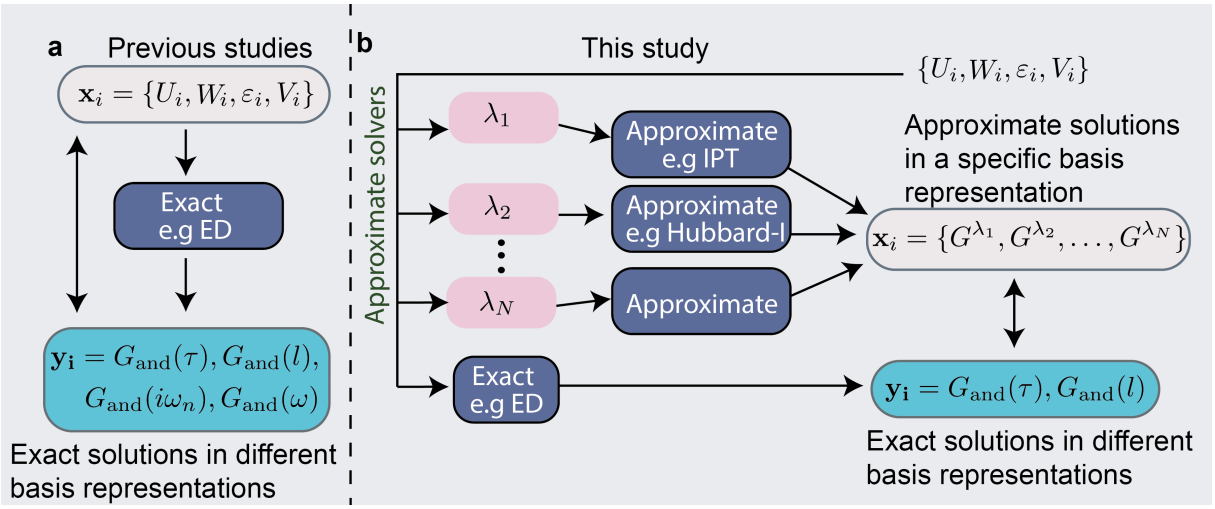

Fig. 6: *Depiction of two approaches for machine learning solutions of the Anderson Impurity Model.* ***a*** *The approach taken by [15] which uses as input features* $\mathbf{x}_i$ *the physical parameters of the AIM* $\{U_i, W_i, \varepsilon_i, V_i\}$ ***b*** *The approach taken here uses approximate solutions, expressed in different bases,* (λ_k)*, of the AIM as the feature vectors.*

We limit our discussion to the single-orbital AIM, which is completely described by the set of parameters $\{U, W, \varepsilon, \beta, V\}$, where U is the Hubbard parameter, W is the half-bandwidth of the bath-states, ε is the impurity on-site energy, β is the inverse temperature and V characterizes the impurity-bath coupling. Furthermore, we can split these contributions into the different components of the AIM, where $\{U, \varepsilon\}$ represent the physics of the impurity while $\{W, V\}$ represent respectively the bath. β is viewed from the perspective of the grand canonical as a fixed parameter of the entire system and the database hence generated at constant temperature. This doesn't preclude generating data for various temperatures, as indeed we'll discuss below the differences that occur in the network as we move from high to low temperature.
In practice, the AIM can absorb inputs in different representations, for instance a set of hamiltonian parameters for ED, or a function in imaginary time for IPT and NCA (the Weiss field). Typically, one might have

- inputs: $\mathbf{x}_i = \{U_i, W_i, \varepsilon_i, V_i\}$
- outputs: $\mathbf{y}_i = \{G(\tau), G(l), G(i\omega_n), G(\omega)\}$,

depending on the choice of basis. The basis choices above are the imaginary-time, Legendre, imaginary-frequency and real-frequency bases, respectively.
Outputs can however be seen as vectors of given length, and in this case, each input vector has a dimension of 4, whereas the output has the number of imaginary-time points (usually $N_\tau > 200$) for the imaginary time Green function, while it is known that the number of Legendre coefficients $N_l < 50$ is more compact [15]. The mapping is illustrated in Fig. 6a. In Ref [15], the authors employ a similar approach for the input features, but attempt to learn the spectral function $A(\omega)$ instead.

2.4 Defining the unknowns: from learning solutions to learning errors

The approach discussed in this lecture is different with regard to the input vectors $\mathbf{x}_i$, where instead we use a combination of approximate solutions to the AIM. We illustrate the idea in Fig. 6b for the i-th AIM instance, which is still characterized by the set $\{U_i, W_i, \varepsilon_i, V_i\}$, but the feature space increases by $N_\lambda N_b - 4$, where N_λ is the number of approximate solvers used, λ refers to the different impurity solver types, and N_b is the number of basis coefficients used.

For example, if we consider using two different approximate solvers $\{\lambda_1, \lambda_2\}$, then the input features for a model would be (let's say for IPT and NCA),

$$\mathbf{x}_i = \left\{G^{\lambda_1}_{\text{and}}(\tau), G^{\lambda_2}_{\text{and}}(\tau)\right\} \quad \text{or} \qquad \mathbf{x}_i = \left\{G^{\lambda_1}_{\text{and}}(l), G^{\lambda_2}_{\text{and}}(l))\right\}, \tag{30}$$

in the imaginary-time or Legendre polynomial bases respectively, and the output vectors are given by,

$$\mathbf{y}_i = G_{\text{and}}(\tau) \quad \text{or} \qquad \mathbf{y}_i = G_{\text{and}}(l). \tag{31}$$

In this case, the length of the feature vector is $2N_b$ and the length of the output vector is N_b. This is a straightforward generalization, we will discuss how this impacts the cost function hereafter.

Moreover, instead of learning the relationship between the input parameters $\{U_i, W_i, \varepsilon_i, V_i\}$ and the exact solution of the AIM, we attempt to learn the *error* between a set of approximate solutions of the AIM and its exact solution, generated using a reference solver such as ED, but equivalently, quantum Monte Carlo can be used as a valid exact benchmark at higher computational costs (we note however that the compute cost for generating a database is not a large concern, as these calculations only need to be performed once and for all).

We then learn the error of a set of approximations, rather than using just one, noting that different approximate AIM solvers have their merits in different parts of the AIM parameter space.

As mentioned in the previous section, IPT is a well-known and successful diagrammatic method for solving quantum many-body problems in the weak-coupling limit, and can also capture some features of the strongly interacting limit. Similarly, the Hubbard-I approach is exact in the weakly hybridized atomic limit, but can qualitatively fail outside of this parameter regimes.

We note that while the weak-coupling expansion has its merits, for scenarios out at strong-coupling it can be qualitatively wrong. To address this, for example, the Non-Crossing Approximation (NCA) is the lowest order strong-coupling perturbative method that sums up all diagrams without crossing hybridization lines.

It is also possible to use basis-truncated approximate ED solutions that use fewer bath sites to represent the Weiss field, which thus provide a more consistent coverage of the parameter AIM space, albeit with larger errors when small numbers of bath fitting parameters are used.

Thus, a natural extension for data driven methods lies with the combination of different approximations, which span a wider range of the AIM parameter space, rather than just using one quantum solver alone.

2.5 Constructing the database

Having established the form of the inputs and their associated outputs, we now discuss the construction of the database. We first deal with inverse temperature, β: unlike the rest of the parameters, the Green function is explicitly dependent on it, so each database is constructed at a specific β. The next parameters to decide are $\{U, W, \varepsilon\}$, all of which are randomly distributed between their extremal values. Therefore, for each instance of an AIM, random samples of each are drawn from the uniform distributions $U \in [U_{\text{min}}, U_{\text{max}}]$, $W \in [W_{\text{min}}, W_{\text{max}}]$ and $\varepsilon \in [\varepsilon_{\text{min}}, \varepsilon_{\text{max}}]$.

Determining the hybridization parameters is slightly more difficult, as regards to how it manifests itself in the impurity solver. The ED solver necessitates a discrete representation of the bath parameters, while this is not true for other impurity solvers, and there is a connection between the lattice bandwidth W and the range of hybridizations that need to be considered in the database.

Furthermore, we want to be able to deal with both discrete and continuous representations of the bath to allow for various approximate solvers. For illustration, we take the Hilbert transform of specific form of the density of states, i.e.,

$$G(z) = \int_{-\infty}^{\infty} \frac{A(\epsilon)}{z - \epsilon}\, d\epsilon \tag{32}$$

with two typical density of states samples being the semi-circular DOS given by

$$A(\epsilon) = \frac{2\sqrt{-\epsilon^2}}{(\pi W)^2} \Theta\big(W - |\epsilon|\big) \tag{33}$$

or the constant DOS, given by,

$$A(\epsilon) = \frac{\Theta\big(W^2 - \epsilon^2\big)}{2W}, \tag{34}$$

where Θ is the Heaviside step-function.

Using either, truncates the limits of integration in Eq. (32) from $-W$ to $+W$, where W is the half-bandwidth, and provides hence the energy scale associated with the hybridization parameters V in the Weiss field ($z = i\omega_n$), the latter being given by,

$$\Delta(i\omega_n) = \sum_{i=1}^{N} \frac{V_i^2}{i\omega_n - \epsilon_i}, \tag{35}$$

where V_i and ϵ_i are the bath parameters, and which become an additional parameter in the database construction.

Specifically, along with the number of samples in the database and the inverse temperature β, the number of bath parameters determines the overall time it takes for the construction of the database. After the number of bath sites is chosen, random samples of each are drawn from the uniform distributions $V \in [V_{\text{min}}, V_{\text{max}}]$ and $\epsilon \in [\epsilon_{\text{min}}, \epsilon_{\text{max}}]$.

To ensure that the discrete representation of the hybridization function retains consistent physical characteristics, its bath parameters are all scaled to the chosen values of W, such that both ϵ_i

and V_i are normalized by it and ϵ_i is centered on its weighted arithmetic mean with respect to V_i^2. Alternatively, it is possible to create discrete representations of the bath by treating ϵ_i and V_i as fit parameters in Eq. (35) to a continuous representation generated from the half-bandwidth W. The next step is to generate the database that will be used for the training of the data-driven model. To do this, each instance of $\{U_i, W_i, \varepsilon_i, \Delta_i(i\omega_n)\}$ is passed to the set of approximate solvers $\{\lambda_1, \ldots, \lambda_N\}$ as well as one exact solver. In this case, the exact solution is obtained by the ED algorithm using a large number of bath sites, generally between 4–6 is enough to ensure a converged solution for the single-site AIM.

3 Training a model solver to solve the Anderson impurity model

We now outline the details of the multivariate maximally connected neural network regression model that is used for the training against the database we have just constructed. As established above, the set of inputs for the model are $\mathbf{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_{N_s}}\}$, where $\mathbf{x_i}$ is a set of different approximate solutions of the AIM, while the outputs are $\mathbf{Y} = \{\mathbf{y_1}, \ldots, \mathbf{y_{N_s}}\}$, where $\mathbf{y_i}$ is an exact solution of the AIM given by ED. For all models trained in this section, they proceed by minimizing the cost function,

$$C(\mathbf{X}, \mathbf{Y}, \boldsymbol{\alpha}) = \frac{1}{N_s} \sum_{j}^{N_s} \big(\mathbf{y}_j - g_{\boldsymbol{\alpha}}(\mathbf{x}_j)\big)^2, \tag{36}$$

with respect to the parameters α to produce a model $g_{\boldsymbol{\alpha}}(x_i) := G^{\mathcal{M}}(x_i)$, where $G^{\mathcal{M}}(x_i)$ is the model Green function of the problem. $G^{\mathcal{M}}(x_i)$ is constructed such that the error between it and the true solution $\mathbf{y}_i$ is minimized, and therefore $G^{\mathcal{M}}(x_i)$ corrects for the error between the approximate solution x_i and the exact one y_i, for all N_s entries in the database. The neural network we use is shown as a schematic in Fig. 7. In the input layer, each neuron evaluates

$$f(\mathbf{x}\cdot\mathbf{w} + \mathbf{b}) = f\Big(\sum_{i,j} w_{ij} G^{\lambda_j}(k_i) + b_1\Big) \tag{37}$$

with $f(\ldots)$ being the activation function of the input layer neurons (colored pink), index i is associated to the feature (i.e. mesh point) and index j indicates the approximate solver used, w_{ij} are the set of neural weights, and $\mathbf{x}_i$ is in general of the format λ_N entries, despite the depiction in Figure 7 that suggests that the number of approximate solvers is 2. This procedure then repeats itself as the values propagate forward through the network such that $f(\mathbf{x}\cdot\mathbf{w} + \mathbf{b})$ of each neuron are used as the inputs for the next layer in the network, until eventually the output layer is reached. As the neural network is being used to solve a regression problem, the output layer applies a linear activation function to its neurons, which does not modify its input data. Therefore, when the output layer is reached the cost function (36) is evaluated for a "mini-batch" of samples, after which the weights throughout the entire network are updated in accordance with the backpropagation method. This procedure is then repeated until $G^{\mathcal{M}}$ is found with weights α that minimize $C(\mathbf{X}, \mathbf{Y}, \boldsymbol{\alpha})$.

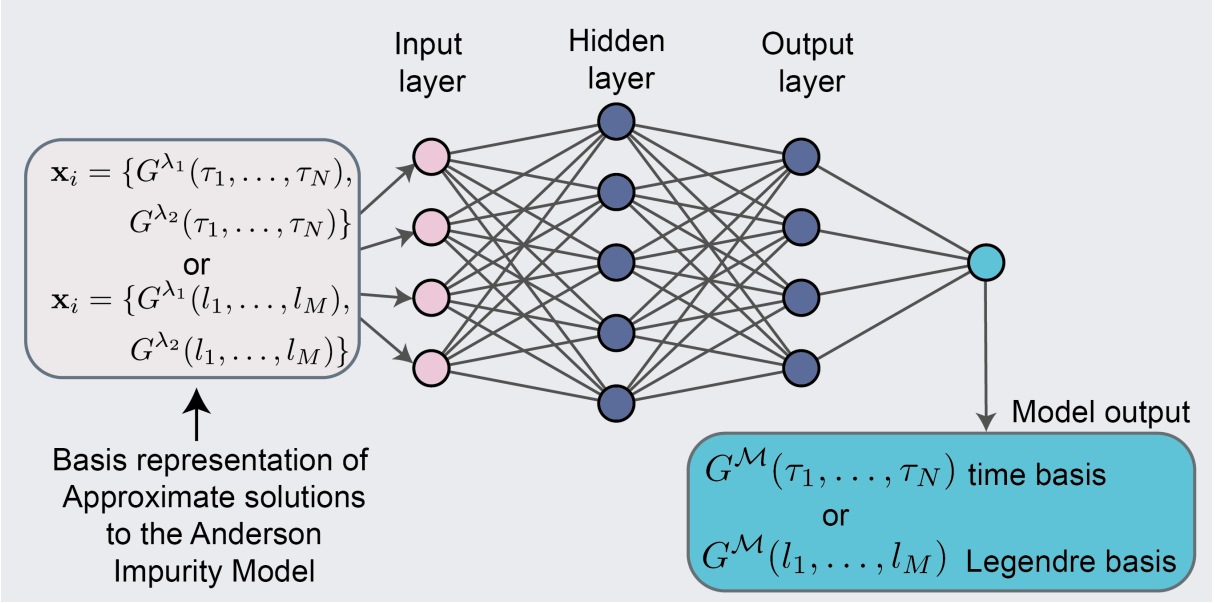

Fig. 7: *Depiction of the maximally connected feed-forward artificial neural network used to generate solutions of the Anderson Impurity Model to produce a model output $G^{\mathcal{M}}(\tau)$ or $G^{\mathcal{M}}(l)$. In the schematic, there is 1 hidden layer with 5 neurons and the output layer has 5 neurons.*

It is important to keep in mind the number of parameters in the model $g_\alpha(x_i)$ so as to avoid potential over-fitting scenarios. For the neural network presented in Fig. 7 the total number of parameters N_α can be determined by the following equation,

$$N_\alpha = \sum_l \left(N_N^l N_N^{l-1} + 1\right), \tag{38}$$

where N_N^l is the number of neurons in layer l. For example, if there are 20 neurons in the input and hidden layer, 100 neurons in the output layer and 200 is the length of the input vector, then the total number of parameters of the network broken down per layer is given by:

$$N_\alpha = \underbrace{20(200{+}1)}_{\text{input layer}} + \underbrace{20(20{+}1)}_{\text{hidden layer}} + \underbrace{100(20{+}1)}_{\text{output layer}} = \underbrace{6540}_{\text{total}}. \tag{39}$$

The value of N_α is pertinent when considering sources of data over-fitting, as it should not exceed to the total number of feature observations in the database. In addition to what determines the number of weights in the neural network, the following series of adjustable parameters have an effect on its performance They are usually referred to as hyperparameters:

- **Learning rate:** step-size update for the weights of the network
- **Mini-batch size:** number of samples after which the neural network weights are updated
- **Epochs:** number of sweeps of the neural network
- **Activation functions:** family of non-linear neuron activation functions, including $\tanh$, elu and relu

- **Cross-validation split:** $\%$-split of the database between training/validation samples
- **Basis functions:** equidistant τ-basis, adaptive τ-basis or Legendre G_l-basis

Typically a hyperparameter grid search is employed over these parameters and by doing so, the model is trained iteratively as the learning steps occur across different values of its parameters. Ultimately it will produce the final optimized value of the cost function for both the training and validation datasets, where the minimum cost function provides a measure for the optimum choice of parameters to be used for the inference process in future DMFT calculations.

3.1 Data processing: symmetry, augmentation and transformation

Once the database of approximate and exact solutions is obtained, and before the data is passed on to the machine learning algorithm for training, there are a number transformation operations that allow the database to be augmented through symmetry operations. Without having to re-run the impurity solver, there are a number of ways to both extend and transform the database in ways that are optimal for learning a model.
For simplicity, we assume a database under consideration is expressed in either the imaginary-time or in the Legendre polynomial basis.
The first symmetry operation makes use of the fact that the Green function can be decomposed into its symmetric and anti-symmetric contributions by decomposing it into the Legendre basis,

$$G^{\mathrm{S}}(\tau) = \sum_{\substack{l\geq 0 \\ \text{even}}} \frac{\sqrt{2l+1}}{\beta} P_l\big(x(\tau)\big)\, G_l \quad \text{and} \qquad G^{\mathrm{AS}}(\tau) = \sum_{\substack{l\geq 0 \\ \text{odd}}} \frac{\sqrt{2l+1}}{\beta} P_l\big(x(\tau)\big)\, G_l, \tag{40}$$

where $G^{\mathrm{S}}(\tau)$ and $G^{\mathrm{AS}}(\tau)$ are respectively the symmetric and anti-symmetric parts of total Green function which give the total Green function when summed, i.e.

$$G(\tau) = G^{\mathrm{S}}(\tau) + G^{\mathrm{AS}}(\tau). \tag{41}$$

In practice, if performed in the τ basis, the latter requires an intermediate step of generating the Legendre coefficients, or reading them in from a database which has them stored already. For the size of the databases dealt with in this lecture (typically less than 40k samples), the latter can be added practically at no additional computational cost. Physically, the symmetric part of the Green function represents the physics at half-filling while the anti-symmetric component encodes the information away from half-filling. This operation need not only be used for the augmentation of the database, it can similarly be used for partitioning it. Specifically, instead of training a model on both the symmetric and anti-symmetric components simultaneously, two separate models can be trained on the symmetric and anti-symmetric components separately, after which they are recombined to produce the total answer in Eq. (41).
The same procedure can be followed in the Legendre basis, where the symmetric part of G_l is encoded in the even coefficients and the anti-symmetric part in its odd ones. For both bases, this operation allows the database to be augmented by a factor of two.

The strategy to learn different features on a given dataset is very much akin to the concepts developed in deep learning: an image would be decomposed in different features with different characteristics, and the neuron model optimized for such.
The second symmetry operation on $G(\tau)$ that we can consider is particle-hole equivalence, i.e., $G^e(\tau) = G^h(\beta-\tau)$ where G^e is the electron Green function and G^h is the hole Green function. Therefore, for every entry in the database that is away from half-filling, the corresponding electron (if hole-type) or hole (if electron-type) Green function can be generated simply by flipping $G(\tau)$. If, however, the database is expressed in the Legendre basis instead, this transformation requires that the odd coefficients be multiplied by -1. We note that for both basis representations, this can double the size of the database.

3.2 Activation function with many-body quantities

In addition to augmenting the database by exploiting symmetry operations, the data must also be transformed into a representation appropriate for how the training data will be manipulated, an in particular for designing a suitable activation function.
Scaling the input and output variables so that they are normalized is a standard technique when preparing data for training algorithms such as neural networks. One example for instance: standard activation functions, as seen in the early chapters of this lecture, are dealing usually with positive signals, so care will be needed to manipulate and transform the Green functions in a suitable format.
Another need for the mapping lies with protecting the weights that are learned in the model from becoming too large or biased towards large input values. Specifically, this is essential for when input variables are the Legendre coefficients, as the Legendre basis has no inherent scale for the coefficients. On the other hand, while an inherent scale exists for $G(\tau)$, i.e., $-1 \leq G(\tau) \leq 0$ when $\tau > 0$, it is also possible to create a family of scaling transformations and test their efficacy throughout the training process. The following scaling transformations work regardless whether the aforementioned symmetrization or augmentation procedures have been followed.
$\mathcal{T}_0$ is the unscaled Green function and each transformation is a function of $\mathcal{T}_0$. For $G(\tau)$ the situation is quite simple, there are only a few transformations that can be done to normalize in between the range $[0, 1]$ or $[-1, 1]$. We note that if $G^{\mathrm{AS}}(\tau)$ is used, i.e., the anti-symmetric part of the Green function, it is important to ensure that the scaling operations do not shift the data out of the scaling range, and so an extra constant shift should be applied in these cases to counteract this behavior. For the Legendre basis, it is clear that the unscaled data is not normalized. Fortunately, by applying a *tanh* function this can readily be achieved. In the example shown, we see the first anti-symmetric component of the Green function, G_1, is scaled to be much closer to G_0 and G_2, the either-side symmetric components. As stated above for the τ basis, the dependence of training the model is also assessed as a function of these transformations.
Moreover, we emphasize that to recover the physical Green function it is necessary to apply the relevant inverse transformation $\mathcal{T}^{-1}$, which are given in Table 1.

a)

	$\mathcal{T}_r$	$\mathcal{T}_r^{-1}$
$\mathcal{T}_0$	$G(\tau)$	$G(\tau)$
$\mathcal{T}_1$	$-G(\tau)$	$-G(\tau)$
$\mathcal{T}_2$	$G(\tau)+1/2$	$G(\tau)-1/2$
$\mathcal{T}_3$	$-2\big(G(\tau)+1/2\big)$	$-G(\tau)/2-1/2$
$\mathcal{T}_4$	$2\big(G(\tau)+1/2\big)$	$G(\tau)/2-1/2$

b)

	$\mathcal{T}_r$	$\mathcal{T}_r^{-1}$
$\mathcal{T}_0$	G_l	G_l
$\mathcal{T}_1$	$\tanh(G_l)$	$\tanh^{-1}(G_l)$
$\mathcal{T}_2$	$-\tanh(G_l)$	$-\tanh^{-1}(G_l)$

Table 1: *Scaling transformations for a): $G(\tau)$ and b): G_l*

3.3 An error correction approach for solving DMFT

Here we present results that pertain to the machine learning framework outlined above. We begin with a discussion of the different aspects of the generated databases, and this is followed by details on the training of an artificial neural network with those generated databases. We conclude by illustrating how the generated data-driven solver is able to capture the Mott transition in the half-filled single-band Hubbard model the using DMFT scheme presented above.

3.4 Database of solutions for the Anderson Impurity Model

Using the procedure outlined in the previous section we generate a database of size $N_s = 10^3$ at inverse temperatures of $\beta = \{1, 2, 5, 10, 20, 50\}$ eV^{-1} over the parameter ranges indicated in Table 2 for discrete sets of bath parameters. The range of temperatures chosen represent the high-temperature and intermediate temperature limits, whereby the features of the Green function are smoother, and hence our choice of the range. Each database entry constitutes a random combination of all parameters in Table 2. The parameters chosen cover a range of physical features, for example the Hubbard U is uniformly randomly sampled over the range $\{1, \ldots, 10\}$, in addition to $W \in \{1, \ldots, 10\}$ and $\varepsilon \in \{-1, \ldots, 1\}$, then the various combinations of U, W, ε result in metals or insulators. Take for instance if $\{U, W, \varepsilon\} = \{8, 2, 0\}$ then the result is insulating, and if $\{U, W, \varepsilon\} = \{2, 8, 0\}$ the result is metallic. In Table 2 we clarify the notation for the approximate solvers ED-[1,2,3]. ED-1 means solving the AIM with one bath site only, and hence results in a truncated approximation to the exact ED solution of the AIM (which in this case uses 4 bath sites). We note that these latter ED solvers are significantly faster than the ED solution obtained at large cost for $N_b > 6$, due to the exponential increase of the Hilbert space. Of course the latter are themselves approximate solutions, similar to IPT or Hubbard-I, and the large error induced by the finite size effects of the bath discretization. We note that interestingly the machine learning framework does act in this respect as a *Hilbert space extrapolation* tool, inferring information on small Hilbert spaces that remains pertinent for larger dimensions.
Furthermore, in Fig. 8 we show the distribution of all chosen parameters for the $10,000$ samples in the database corresponding to $\beta = 20$ eV^{-1}. As expected, $\{U, W, \varepsilon\}$ are distributed evenly, $N_b = 4$ is constant as the number of bath-sites is not changed, and $\{\epsilon_i, V_i\}$ are chosen by normalizing to W. While the illustrated database is not the only one that could be considered, it is not a special choice. For all other databases we analyzed, the distribution of parameters behaves similarly to the $\beta = 20$ eV^{-1} case presented.

U (eV)	$\{1, \ldots, 10\}$
N_{bath}, ϵ_i, V_i	4
W (eV)	$\{1, \ldots, 10\}$
ε	$\{-1, \ldots, 1\}$
β (eV^{-1})	$\{1, 2, 5, 10, 20, 50\}$
N_{samples}	10,000
$\mathcal{S}$	Hubbard-I, IPT, NCA, ED-[1,2,3]

Table 2: *Parameter selection for the database of AIM solutions shown in Fig. 8 where* $\{p_i, \ldots, p_f\}$ *denotes that a parameter is randomly selected from this interval* $[p_i, p_f]$. U *is the Hubbard interaction,* N_{bath} *stands for the number of bath sites,* W *is the Half-bandwidth,* ε *is the electron on-site energy,* β *is the inverse temperature,* $N_{samples}$ *is the number of database entries, and* $\mathcal{S}$ *denotes the total ensemble of approximate quantum solvers used in the ML approach. ED-[1,2,3] denotes the exact diagonalization solver with respectively* $1, 2, 3$ *bath sites.*

We review the strength and weaknesses of the Hubbard-I, IPT and NCA solvers for representative samples of the database against the corresponding exact diagonalization results. The latter provide valid approximations of the AIM in different limits (Hubbard-I and IPT are good in the weakly hybridized limit, NCA is a good approximation for stronger interactions). In general, the Hubbard-I, IPT and NCA solvers are however in quantitative and qualitative disagreement.

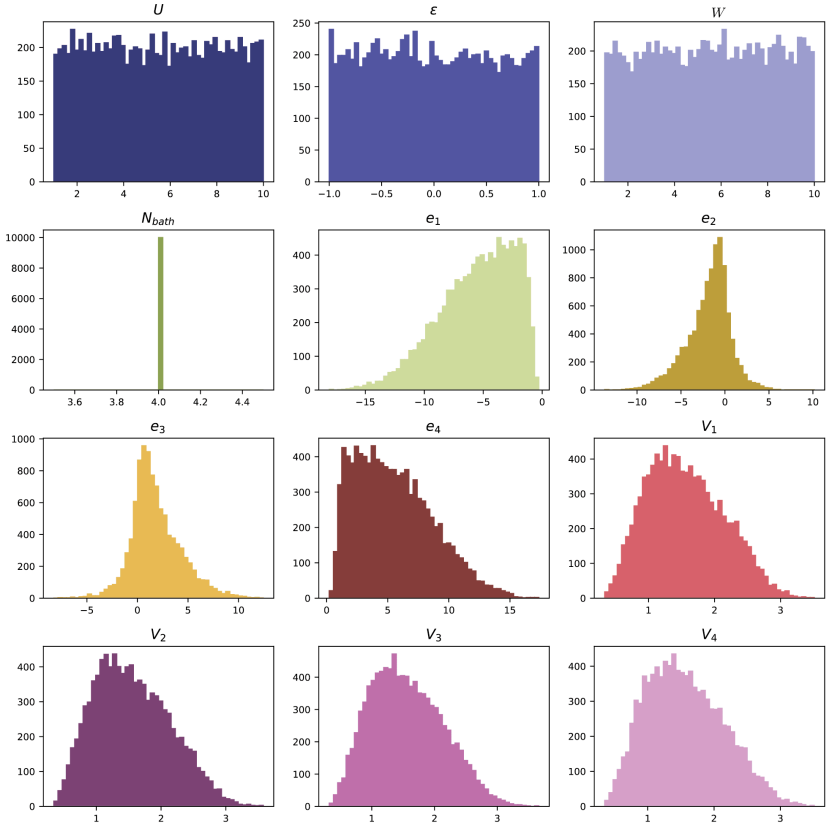

Fig. 8: *Typical statistical distribution of the Anderson Impurity Model parameter space for a database for* $\beta = 20\ eV^{-1}$, $N_{bath} = 4$ *with 10,000 entries.*

Hyperparameter	Range	Optima
η	$[0.0001, 0.0002, 0.001, 0.01]$	0.0002
Mini-batch size	$\{4, 8, 16, 32, 64\}$	8
Optimizer	{Adamax, Nadam}	Nadam
Activation functions	{elu, relu, *tanh* }	*tanh*
Hidden Layers	$\{1, 2, 3, 4\}$	1

Table 3: *Hyperparameter grid-search over the neural network parameters with a fixed number epochs = 20 and* $\beta = 1$ *eV*$^{-1}$

For example, the Hubbard-I solver indicates for highly hybridized AIM an insulating solution when in reality the system is metallic. Nevertheless, we emphasize that this behavior is expected and welcome, since the end-goal is to systematically correct for the error between the approximate and exact solutions.

3.5 A Neural Network Impurity Solver

The very first step to training a machine learning model is the hyperparameter grid-search over its independent parameters using tensorflow [24]. Specifically, for our neural network we coarse-grain the number of epochs to 20, set the inverse temperature to $\beta = 1$ eV^{-1} and scan across all combinations of parameters in Table 3. Ultimately, 401 different neural networks are trained and the combination of parameters with the minimum cost function $\sim 10^{-6}$ is given by that combination of parameters shown in the "Optima" column of Table 3. Additional fixed parameters in the grid search are: evenly spaced time grid, Hubbard-I, IPT and NCA solvers as inputs $\mathbf{x}_i$ to the neural network as they all require minimal computational resources in comparison to the ED methods, no data augmentation, and the $\mathcal{T}_4$ imaginary-time transformation from Table 1. It is noteworthy that either increasing the complexity of the network, i.e., increasing its depth beyond 1, or increasing the learning rate to an order beyond 10^{-2} has detrimental effects on the minimization of the cost function. Practically, it would be computationally prohibitive to perform this grid search for every β and their additional free-parameters. In what follows, all networks use the optimal values as specified in Table 3 and use an $80/20$ cross-validation split, i.e., 80% training data and 20% validation data.

We propose a collection of data scaling transformations of the input and output data which improve the training of the neural network in the imaginary-time and Legendre bases. Fig. 9 presents the validation loss for these scenarios, for $\beta = 1$ eV^{-1}. For the Legendre basis the effect of data transformations is quite significant, as shown in Fig. 9. Here we see that by applying a *tanh*-type Legendre transformations, the final value of the loss can be improved by at least 2 orders of magnitude, reduced from 10^{-4} to 10^{-6}. We expect the effect of this transformation to be enhanced for larger values of β (lower temperatures), where the range of G_l can greatly exceed the value of unity, and therefore necessitates the application of an appropriate data transformation. At higher temperatures (*i.e* lower β), the Legendre coefficients are often bounded close to unity, and so the neural network is less sensitive to the untransformed input as compared to lower temperatures.

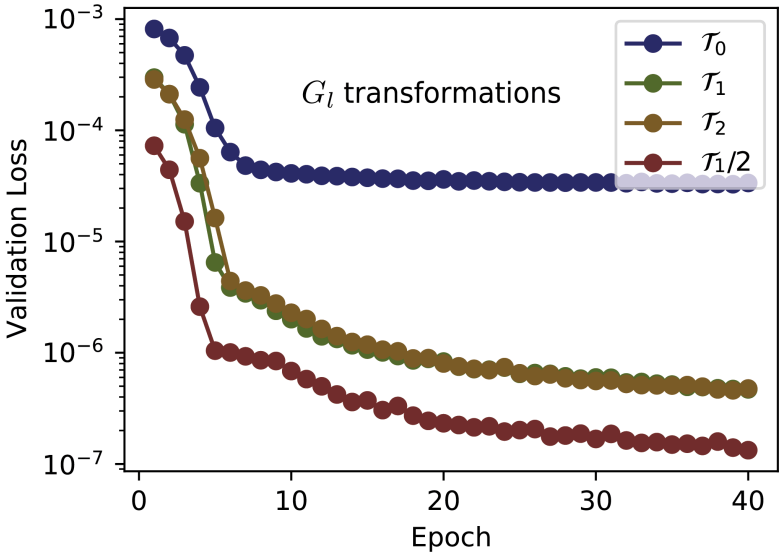

Fig. 9: *Validation loss for the Legendre basis transformations at $\beta = 1$ eV^{-1}. The transformations applied on the database lead to different figures of merit for the network predictions.*

In Figure 10 we show the value of the cost function when trained in the Legendre basis using the $\mathcal{T}_1/2$ data transformation. We observe for the training in the Legendre basis that higher temperatures are more amenable to the training procedure and that including more approximate solvers increases accuracy of the final validation loss. Therefore, we see that by executing suitable basis transformations which are supplemented by a multitude of different approximate solvers, the accuracy of the overall predictive quality of the neural network can be improved. We note that production of high quality data on larger values of β requires a larger number of imaginary time slices or Legendre coefficients.

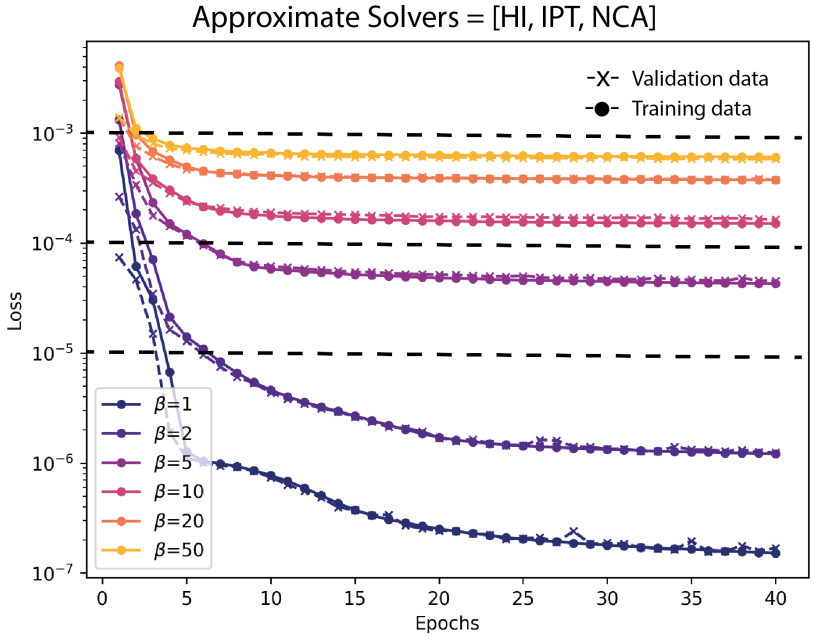

Fig. 10: *Cost function for the Legendre mesh for different inverse temperatures β using as input the approximate solvers Hubbard-I, IPT and NCA.*

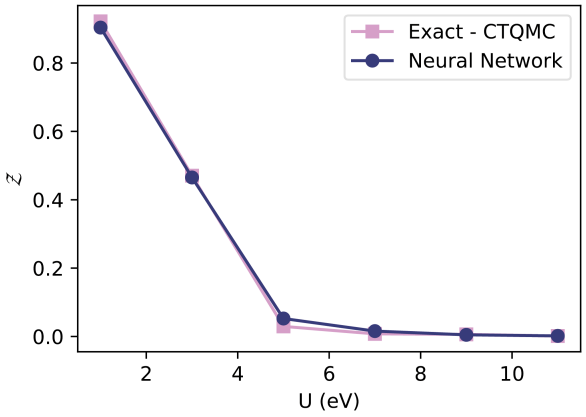

Fig. 11: *CTQMC and Neural Network solvers used for a DMFT prediction of the quasiparticle weight $\mathcal{Z}$ as a function of U at $\beta = 20$ eV^{-1} and $W = 1.0$ eV for the single-band half-filled Hubbard model on a Bethe lattice.*

3.6 A data-driven approach to the Mott transition

The motivation behind developing the data-driven impurity solver is to alleviate DMFT calculations from the intensive computational burden imposed by Exact Diagonalization and Monte Carlo methods. In Fig. 11 we illustrate how the neural network solver, used in a DMFT calculation, can predict the Mott transition at $\beta = 20$ eV^{-1}, $W = 1.0$ eV (at half-filling). This is compared with the equivalent exact CTQMC results.
For each value of U, both solvers are run for 30 iterations to a self-consistent solution. As U is increased the Mott transition occurs at $U/D \approx 4$–6, consistent with other calculations in the literature [1], up to a factor of 2 due to the choice of $D = 2eV$. We emphasize that the network uses approximate solutions as its input during its cycle, for which it predicts the error-free corrected output instantly. By contrast the CTQMC has to be run long enough to mitigate its statistical error bars. This proof-of-concept calculation highlights the power of the data-driven method for a prototypical strongly correlated system, where the solver runs nearly instantaneously, without any significant overheads.

4 Conclusion and code availability

We reviewed neural networks as a data-driven framework that can readily be trained for providing solutions of the Anderson impurity model. This provides an impurity solver capable of capturing the Mott transition using DMFT for the Hubbard model. So far this approach remains robust at higher temperatures, using approximate solutions results in consistently reliable results. We anticipate that improved results at lower temperatures could be attained by extending this method to larger databases or more compact representations of the Green function.
The code discussed in these notes, coined *Data driven Dynamical Mean Field Theory* (D^{3}MFT) is available on GitHub at *http://github.com/zelong-zhao/d3mft*. For the installation of this package, please make sure that you have the Anaconda manager installed on your system, then simply run *./install.sh d3mft*. Once installed, there are different examples which can be run.

References

[1] A. Georges, G. Kotliar, W. Krauth, and M.J. Rozenberg, Rev. Mod. Phys. **68**, 13 (1996)

[2] G. Kotliar, S.Y. Savrasov, K. Haule, V.S. Oudovenko, O. Parcollet, and C.A. Marianetti, Rev. Mod. Phys. **78**, 865 (2006)

[3] A.N. Rubtsov, V.V. Savkin, and A.I. Lichtenstein, Phys. Rev. B **72**, 035122 (2005)

[4] A.K. Mitchell and L. Fritz, Phys. Rev. B **88**, 075104 (2013)

[5] C.A. Perroni, H. Ishida, and A. Liebsch, Phys. Rev. B **75**, 045125 (2007)

[6] M Ceperley and B.J. Alder, Phys. Rev. Lett. **45**, 566 (1980)

[7] J. Behler and M. Parrinello, Phys. Rev. Lett. **98**, 146401 (2007)

[8] Z. Li, J.R. Kermode, and A. De Vita, Phys. Rev. Lett. **114**, 096405 (2015)

[9] A.P. Bartok, M.C. Payne and G. Csanyi, Phys. Rev. Lett. **104**, 136403 (2010)

[10] J. Liu, Y. Qi and L. Fu, Phys. Rev. B **95**, 041101 (2017)

[11] L. Huang and L. Wang, Phys. Rev. B **95**, 035105 (2017)

[12] H. Shen, J. Liu and L. Fu, Phys. Rev. B **97**, 205140 (2018)

[13] P. Broecker, J. Carrasquilla and S. Trebst, Sci. Rep. **7**, 8823 (2017)

[14] L.F. Arsenault, R. Neuberg and A.J. Millis, Inverse Problems **33**, 115007 (2017)

[15] L.F. Arsenault, A. Lopez-Bezanilla, and A.J. Millis, Phys. Rev. B **90**, 155136 (2014)

[16] E.J. Sturm, M.R. Carbone, and R.M. Konik, Phys. Rev. B **103**, 245118 (2021)

[17] N. Walker, S. Kellar, and K.M. Tam, arXiv:2008.12331

[18] H. Hafermann, C. Jung, and A.I. Lichtenstein, Euro. Phys. Lett. **85**, 27007 (2009)

[19] E. Sheridan, C. Rhodes, F. Jamet, I. Rungger, and C. Weber, Phys. Rev. B **104**, 205120 (2021)

[20] H. Jung, R. Silva and M. Han, World Electric Vehicle Journal **9**, 46 (2018)

[21] F Rosenblatt, Psychological Review, **65**, 386 (1958)

[22] C.M. Bishop: *Pattern Recognition and Machine Learning, Information Science and Statistics* (Springer, 2006)

[23] M. Caffarel and W. Krauth, Phys. Rev. Lett. **72**, 1545 (1994)

[24] M. Abadi et al.: *TensorFlow: Large-scale machine learning on heterogeneous systems* `http://tensorflow.org` (2015)