This is a repository copy of *Extrapolating from neural network models: a cautionary tale*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/id/eprint/180478/

Version: Published Version

Article:
Pastore, Alessandro orcid.org/0000-0003-3354-6432 and Carnini, Marco (2021) Extrapolating from neural network models: a cautionary tale. Journal of Physics G: Nuclear Physics. 084001. ISSN: 0305-4616

https://doi.org/10.1088/1361-6471/abf08a

Reuse
This article is distributed under the terms of the Creative Commons Attribution (CC BY) licence. This licence allows you to distribute, remix, tweak, and build upon the work, even commercially, as long as you credit the authors for the original work. More information and the full terms of the licence here: https://creativecommons.org/licenses/

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

Journal of Physics G: Nuclear and Particle Physics

PAPER • **OPEN ACCESS**

Extrapolating from neural network models: a cautionary tale

To cite this article: A Pastore and M Carnini 2021 *J. Phys. G: Nucl. Part. Phys.* **48** 084001

View the article online for updates and enhancements.

You may also like

- Extrapolation schemes of key comparison results in gas analysis
Adriaan M H van der Veen and Heleen Meuzelaar
- Semiclassical *p*-branes in hyperbolic space
Rodrigo de León Ardón
- The distributional stress–energy quadrupole
Jonathan Gratus, Paolo Pinto and Spyridon Talaganis

This content was downloaded from IP address 79.76.223.82 on 16/11/2021 at 09:54

OPEN ACCESS
IOP Publishing
Journal of Physics G: Nuclear and Particle Physics
J. Phys. G: Nucl. Part. Phys. **48** (2021) 084001 (17pp)
https://doi.org/10.1088/1361-6471/abf08a

Extrapolating from neural network models: a cautionary tale

A Pastore[1,*] **and M Carnini**[2]

[1] Department of Physics, University of York, Heslington, York, Y010 5DD, United Kingdom
[2] Features Analytics, Rue de Charleroi 2, 1400 Nivelles, Belgium

E-mail: alessandro.pastore@york.ac.uk and marco.carnini@features-analytics.com

Received 11 December 2020, revised 3 March 2021
Accepted for publication 19 March 2021
Published 21 June 2021

Abstract

We present three different methods to estimate error bars on the predictions made using a neural network (NN). All of them represent lower bounds for the extrapolation errors. At first, we illustrate the methods through a simple toy model, then, we apply them to some realistic case related to nuclear masses. By using theoretical data simulated either with a liquid-drop model or a Skyrme energy density functional, we benchmark the extrapolation performance of the NN in regions of the Segrè chart far away from the ones used for the training and validation. Finally, we discuss how error bars can help identifying when the extrapolation becomes too uncertain and thus not reliable.

Keywords: neural network, error bars, nuclear binding energies

S Supplementary material for this article is available online

(Some figures may appear in colour only in the online journal)

1. Introduction

Neural networks [1] (NNs) are powerful tools that are widely used in several domains of science. Within the nuclear physics community, several groups have started investigating the NN as a tool to improve current models [2, 3] in predicting specific observables like nuclear masses [4–9] and radii [10], or as intermediate tool to avoid time-consuming calculations [11]. The domain of application is so vast since NN are *universal* approximators [12, 13]: any continuous function can be approximated by an NN with a single hidden layer having a sufficiently large

*Author to whom any correspondence should be addressed.

Original content from this work may be used under the terms of the Creative Commons Attribution 4.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

number of neurons. Each neuron forming the NN contains some adjustable (hyper-)parameters (weights and biases) that are determined in order to minimise a given *objective* or *loss* function, typically the mean squared error.

Once the architecture of the network is fixed, i.e. the number of layers, nodes and the connection patterns, the critical aspect is to optimise the values of the weights and biases. To some extent, this is the same procedure commonly used in nuclear physics to adjust model parameters [14, 15]. As a consequence, it is important to equip the NN with a reasonable estimate of the error bars to help assessing the quality of the results.

As discussed in reference [16], estimating theoretical errors is not an easy task since many factors could contribute to them. In particular, one can model the prediction error in terms of two main components: a *statistical* and a *systematic* one. The first arises from the optimisation procedure and it can be evaluated using specific statistical tools, while the second is usually unknown.

The standard strategy used to estimate error bars is based on the covariance matrix and the first derivative of the model in parameter space. See references [16, 17] for more details. This procedure can not be applied to a typical NN for a very simple reason: the number of parameters is very large, typically in the range of thousands. Since the NN is non-linear in parameter space [18], it follows that the covariance matrix needs to be evaluated by performing numerical derivatives in the parameter space. Due to the possible numerical issues discussed in reference [17], we prefer not to explore this method. A first attempt to include error bars within NN can be found in reference [19].

In the present article, we study three different methods to estimate error bars that do not require major modification to the existing Python functions. To this purpose, we train a series of NNs using nuclear mass data derived from existing nuclear models: the liquid-drop (LD) [20] and the Skyrme nuclear energy density functional (NEDF) [21]. The idea is very simple: guided by the current knowledge of nuclear masses [22], we separate our theoretical masses in two sets: one corresponding to the measured ones ($\approx$2400 nuclei) and the other corresponding to the extrapolated region. The role of experimental binding energies is to guide us in determining in which group a given nucleus should classified. We then use the first set for the training/validation of the NN and the second one to benchmark the extrapolation. This mimics the current situation where in the scientific literature several authors try to extrapolate models in regions where no data are available. The choice of using data generated through models and not experimental values is dictated by the desire of working on problems for which the values of the mass are available, to assess the validity of the error bars. We are aware that the trained NN will not produce meaningful estimations for nuclei. The best that can be achieved is to approximate the explicit formula of the model (say, LD) through the NN.

Our goal is to show that even this simple task is challenging, and that extrapolation can be at best attempted close to the region of the known nuclear masses. A more general discussion on the extrapolation properties of NN can be found in reference [23].

The article has been structured to be used as a guide to navigate the associated Jupiter notebooks made available as supplementary material (https://stacks.iop.org/JPG/48/084001/mmedia). The article is organised as follows: in section 2 we briefly introduce the NN and its error estimate and we apply it to a simple toy-model; in section 3 we present the nuclear models we use to obtain the data-set; in section 4 we discuss the various methods to estimate error bars applied to realistic cases. Finally, we present our conclusions in section 5.

2. What is a neural network?

A feedforward NN [1] is an ensemble of elementary units called neurons, organized into an input layer, an output layer, and generally one more intermediate or *hidden* layer. Processing occurs exclusively in the forward direction, from input to output. Individual neurons are represented mathematically by an *activation* function f (corresponding to the biological action potential), which takes the form of a sum of weighted inputs from neurons in the preceding layer to produce an output y that serves as input for neurons of the succeeding layer. Explicitly

$$y = f\left(\mathbf{x} \cdot \mathbf{w} + b\right), \tag{1}$$

where $\mathbf{x}$ is the input vector to a given neuron, $\mathbf{w}$ is the corresponding vector formed from the layer's connection matrix, and b is the neuron's bias. The values of $\mathbf{w}$ and b are not known and they need to be determined by training the network. The goal of the training process is to minimize the mean square error (MSE)

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(Y_i - \hat{Y}_i\right)^2, \tag{2}$$

where N is the number of observations used to train the network, $\hat{Y}_i$ is the prediction of the network for observation i while Y_i is the actual value for observation i. Notice that Y can be the result of an experiment or of a theoretical calculation.

It is well known that a feed-forward NN with a non-polynomial activation function and a single hidden layer can approximate *any* continuous function [24]. The quality of the approximation function will depend on the number of hidden neurons, and also on the training procedure adopted [25], but even on the initialisation of the weights and biases [26]. The theorem given in reference [24] is independent on the details of the training procedure. However, the time required to train a large NN can be prohibitive and the scarcity of data can limit the applications of NN. It is thus important to train the NN using the best *features* (i.e. input-layer variables) in order to maximise the quantity of information one can extract from the data. See discussion in reference [27] for more details.

To better illustrate such a concept, we present a simple toy-model. The calculations were performed using a Jupiter notebook that is provided as a supplementary material. The application of NN to the nuclear case starts at section 3.

2.1. Fitting a parabola

We generate 200 points in the interval $x \in [0, 1)$ and evaluate the function $Y(x) = x^2$. We split the data set into a *training* and *validation* as 80% and 20% of the data. For the purpose of this example, we build a single layer NN using eight neurons. We selected the rectified linear (ReLu) $f(x) = \max(0, x)$ [28, 29] as activation function. This is a quite popular choice since it allows to train networks with several layers without incurring in gradient vanishing problems. The weights are initialised using a *glorot uniform* [30], i.e. they are drawn from a truncated uniform distribution defined over the interval $\left[-\sqrt{\frac{6}{n_i+n_{i+1}}}, \sqrt{\frac{6}{n_i+n_{i+1}}}\right]$. n_i indicates the size of the layer i. See reference [30] for more details.

The glorot uniform is the default option for the initialisation of the weights for the dense function in Keras. A different initialisation procedure could be used instead: for instance see discussion in reference [31]. Throughout this paper, the optimisation of the weights is done

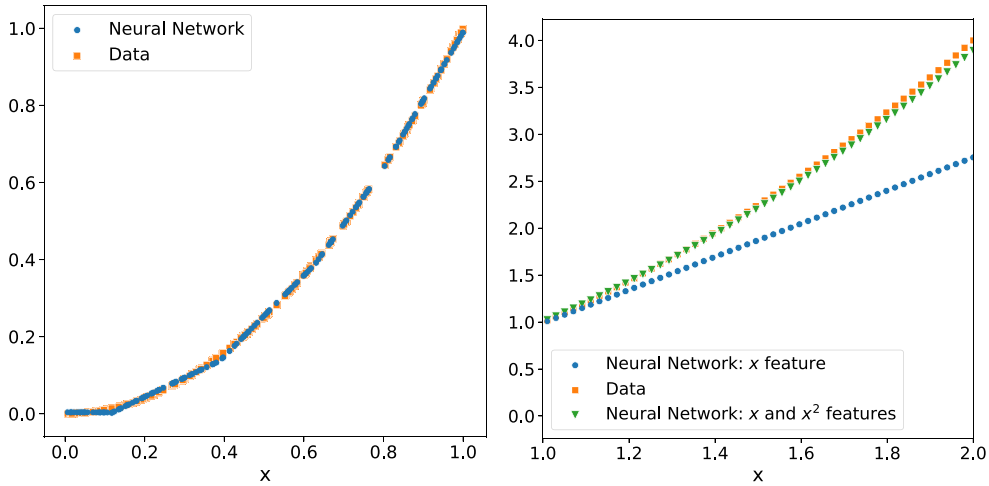

Figure 1. Left: comparison between NN interpolation (circles) and the original data points of the model (squares). Right: comparison between NN extrapolations with and without additional features (lower triangles and circles, respectively) and data points extracted from the original function $Y(x) = x^2$ (squares). See text for details.

using the Adam optimiser [32]. This is quite a common choice within the machine learning community.

After training the NN over 2000 epochs, we obtain the result given in the left panel of figure 1. We observe that the network is able to grasp the structure of the data. The mean square error, equation (2), on the training set is $\approx 5 \times 10^{-6}$. For this particular case, the MSE on the validation set is also very similar showing that the NN provides a very good approximation of the data. See the supplementary materials for additional details.

However, it is worth investigating what happens when we extrapolate using the NN in a region of input space where there are no training data. In the right panel of figure 1, we illustrate the extrapolation of the NN in the interval $x \in [1, 2]$ and we compare it with the *true* model $Y(x) = x^2$. We see that the extrapolation quickly deteriorates and eventually the difference between the ground truth and the NN predictions quickly increases. In particular, we notice that the NN is not able to capture the quadratic behaviour of the data and we see a clear linear dependence in the region of the extrapolation—a side effect of choosing ReLU as an activation function.

NN can *learn* any type of structure in the data [24], but we can help the network by providing extra information at the input, exactly in the same way as we did in reference [27]. In data science lexicon, this is called *feature engineering*. See reference [33] for additional details. The role of feature engineering is to improve the predictions obtained with the NN without increasing the number of data or changing the architecture, notably the number of hidden neurons or hidden layers.

For this example, we train a new NN using exactly the same layout, but now including as input data both x and x^2. In this case, we know the *exact* structure of the toy model, but in a realistic case one should explore various possibilities.

We stress that we are not changing the data, but we are simply making a transformation to highlight possible patterns in the training set. In other terms, we are only adapting the data *representation* to the algorithm (NN) that we chose.

In the right panel of figure 1, we compare the trained NN with the additional feature x^2 (triangles) to the simpler NN (full circles). In the region $x \in [0, 1]$ both NN do very well, but the one using additional features clearly behaves way better in the interval $x \in (1, 2]$.

Finding the most relevant feature to improve the quality of the network during the training process is not an easy task. It is thus important to assess the quality of the NN, by defining error bars that may guide us in evaluating the quality of an extrapolation in a realistic case, where we do not know the structure of the *exact* model. Finally, it is worth mentioning that adding *features* that are not relevant for the model could actually lead to a deterioration of the results. See for example the discussion in reference [34].

2.2. Error bars

Within scientific literature there is no consensus on how to estimate error bars for NN. Assuming that the errors are normal, a very simple approach to estimate error bars is represented by setting the error bar to 1σ [18], equal to the global root mean square (RMS) of the model on the validation set, as done in reference [9]. In this case the error bars are too small to be visible (they are actually hidden by the size of the point we chose).

The standard approach based on the covariance matrix [16] is not suitable due to the typical large number of parameters and the clear difficulties in performing numerical derivatives in parameter space [17, 35]. In the following, we investigate three possible methods using the example illustrated in section 2.1. For simplicity we consider x as the only feature to train the network.

A possible alternative to the approaches presented in this article is represented by Bayesian neural networks (BNNs): in this case instead of obtaining a point estimate, the BNN produces a posterior distribution of a given quantity [36]. Given the complexity of the topic, we prefer not to discuss BNNs in the current article and leave them for a future work. For more details on BNNs we refer the reader to reference [37].

2.2.1. Epoch averaging. During the training of an NN, it is possible to store its coefficients at fixed values of epochs during the training [38]. The approach was introduced independently by Polyak [39, 40] and Ruppert [41], and it is known as Polyak or Polyak–Ruppert averaging. In the case of this toy model, there is no variability, since the coefficients converge to their final result after few epochs and the gradient vanishes (see the supplementary material). In a *realistic* case, as shown in figure 3, one observes that the MSE as a function of epochs decreases until it reaches a *plateau* where it starts fluctuating since the gradient does not reach a stable minimum. This means that the coefficients of the network are still slightly varying as a function of the number of epochs. By storing them, for example, every 1000 epochs, we effectively create different networks with similar MSE. Having access to the different networks, we define the error bar in a statistical way as the interval where middle 68% of predictions lie. This is done under the assumption that the errors follow a Gaussian distribution: the extrapolated value is the mean and the error bar is the standard deviation.

The main advantage of this method is that we do not need to train additional networks and thus it is a remarkable gain in central processing unit (CPU)/graphics processing unit (GPU) time. The downside is that all these networks are not independent from one another, and they typically manifest a strong degree of correlation. This may lead to an underestimation of the error bars width. The additional downside is that if the gradient vanishes during the training, the NN reaches a stable configuration and as such averaging over successive epochs does not introduce any variability. This would also lead to an underestimation of the error bars.

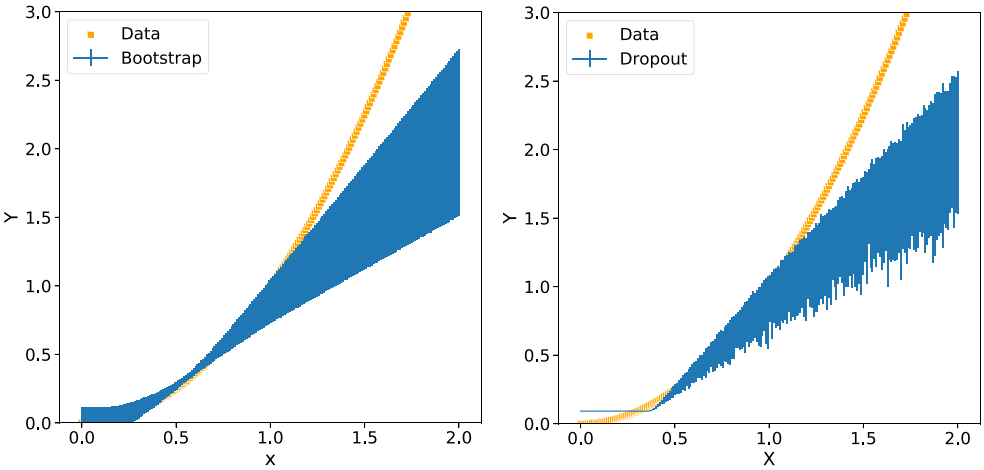

Figure 2. Left: comparison between NN using bootstrap (solid line) and the original data (squares). Right: comparison between NN using dropout (solid line) and the original data (squares). See text for details.

2.2.2. Bootstrap. A very simple alternative to the epoch-averaging is based on bootstrap [42, 43]. Since the training of a network is essentially a non-linear fit, we can explore the landscape of parameter space by using slightly different training sets. To this purpose, given the data, we create 100 sets of validation/training sets, by random sampling the original one. We have tested that increasing the number of samples does not change our results.

For each of them, we train an NN with the same architecture. We see that each individual network will be trained only on a fraction of the total data (here 80%), but the full ensemble will be trained over *all* the data. As a consequence, using bootstrap, we maximise the information contained in the data. Having access to 100 networks, we average them out and define an error bar as the region where 68% of the curves lie, while the expected prediction is obtained by simply averaging over the outcomes of all NNs.

In the left plot of figure 2, we show the resulting prediction obtained using the bootstrap method: we see that the error bars are very small in the region [0, 1] while they grow when x approaches the limits of the data set. Beyond $x = 1$, the error bars grow remarkably fast, as expected due to the lack of information in this region of space.

The error bars obtained with bootstrap strongly depend on the variability of the predictions. By investigating in more detail the left plot of figure 2, we observe that the *true* model falls within the error bars only up to $x \approx 1.25$. To go beyond this point one should use larger error bars by taking, for example, two standard deviations, but one clearly notices that this makes the prediction less and less relevant given the size of the error bars.

2.2.3. Dropout. Finally, we consider the dropout method to estimate the error bars. This is a technique that has been introduced to avoid over-fitting and to improve predictions. The idea is very simple [44]: we train the network over the training set and we randomly switch off some neurons. In this simple toy model we drop one neuron each time. We obtained 100 distinct predictions with the trained NN, switching off (on average) one neuron randomly each time. In this way we produce 100 predictions and we define the error bar, as in the bootstrap case, as the region where 68% of the predictions lie. The result is illustrated in the right panel of

figure 2. As in the previous case, we have tested that using a larger set of samples does not change our results.

By comparing the results for bootstrap and dropout in figure 2, we see that the dropout gives very similar results to the bootstrap case. The error bars estimated in this way contain the *true* model up to $x \approx 1.25$. As discussed in the previous Bootstrap case, one could consider larger error bars by taking two standard deviations, but this makes the extrapolation less and less relevant.

We now move to some realistic nuclear data to continue testing the three methods presented here to evaluate error bars.

3. Nuclear masses

Currently [22], more than 2400 nuclear masses have been experimentally measured with very high degree of accuracy. The exact knowledge of nuclear binding energies play a crucial role in several physical scenarios as for example r-process nucleosynthesis [45] or in the determination of the chemical composition of the crust of a neutron star [46]. Since NN have been recently applied to perform extrapolations of nuclear masses [6–9] in regions where no experimental data are still available, we find it very important to provide a reliable estimate of the error bars to help evaluating the quality of such results.

As done in the previous section, we validate the quality of the extrapolation against a *closed-form* model. To this purpose, instead of using experimental masses, we use binding energies calculated from a nuclear model. The synthetic data are generated using a LD model [43] and the NEDF via the Skyrme SLy4 parametrisation [47, 48]. Both LD and SLy4 give a reasonable reproduction of nuclear binding energies, with a RMS of few MeV. Other mass-models with improved accuracy are available within the literature [49–52], but—for the present calculation—we are only interested in considering two categories of models: linear and non-linear [18], just to check if the performances of the NN are impacted by such a choice.

Both LD and SLy4 predict the existence of way more nuclei than the one experimentally observed [22] and they allow us to benchmark against NN predictions along several complete isotopic chains.

To be as realistic as possible, we define for the training set of the NN all the measured isotopes given in reference [53], but using the values of binding energies calculated by the models. We stress that at this stage, we are not interested in reproducing as accurately as possible experimental data, but to validate the approach for extrapolations based on NN.

We build an NN formed by three layers having composition 16–8–16 with neurons densely connected, and adopting a ReLu activation function. No particular effort was dedicated to fine tune and optimise the architecture, except for fixing reasonable defaults (see for example [29]). Following reference [54], the NN is directly trained on total binding energies per particle to avoid any additional bias introduced by the model itself.

3.1. Liquid drop data

The LD model expresses the nuclear binding energy, B, as a sum of five different terms depending uniquely on proton (Z) and neutron (N) number as

$$\frac{B^{\mathrm{LD}}(N,Z)}{A} = a_v - a_s A^{-1/3} - a_c \frac{Z(Z-1)}{A^{4/3}} - a_a \frac{(N-Z)^2}{A^2} - \delta \frac{\mathrm{mod}(Z,2) + \mathrm{mod}(N,2) - 1}{A^{3/2}} \tag{3}$$

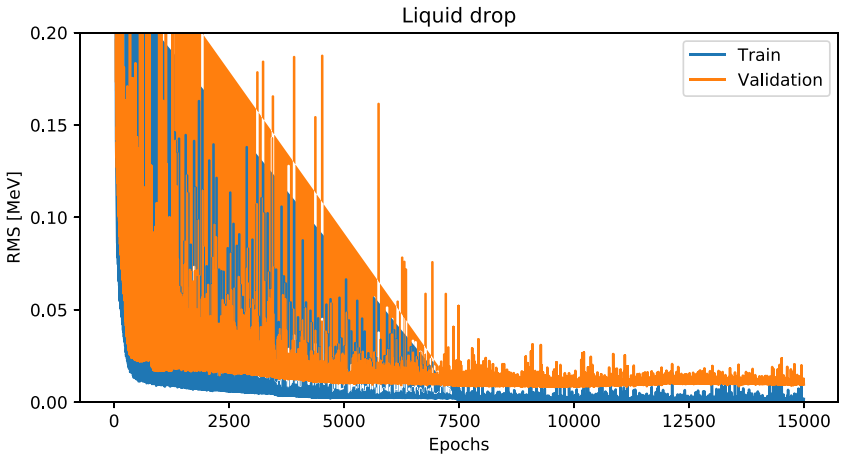

Figure 3. Evolution of the RMS, equation (2), as a function of the epochs used for training the NN and based on LD data. See text for details.

where $A = N + Z$ and the coefficients $a_v, a_s, \ldots$ have been adjusted in reference [43]. The features of the model are quite simple and one could use them to directly train the network [27]. However, here we consider no *a priori* knowledge of the model and we use only N and Z as features.

In figure 3, we show the evolution of the RMS (expressed in MeV) as a function of the epochs, for both training and validation sets. We used as a label the energy per particle B/A given by equation (3). The motivation for our simple choice of the network can be seen here: after few thousands of epochs, the network has reached a *plateau*, where the gradient is small. The fast convergence is a necessary ingredient for our successive analysis on error bars.

The final RMS on the training set is $\sigma_{\rm tr} \approx 50$ keV, while on the validation is $\sigma_{\rm val} \approx 100$ keV. The accuracy is roughly of the same order of magnitude of the LD with respect to experimental nuclear data. By training a second NN on the residuals it would be probably possible to further reduce the RMS [54], but this is not the goal of the present discussion. We want to stress that the NN trained here is probably not the best one we can build out of the data, but a reasonable tool that we can use for our analysis on error bars.

Having trained the NN, we define the residuals as the difference between the $B^{\rm LD}/A$ and the binding energy per particle as calculated via the NN $B^{\rm NN}/A$. In figure 4, we compare the evolution of the residuals as a function of the mass number A for two isotopic chains: calcium and lead. The choice of the isotopic chains is somehow arbitrary: we picked those to illustrate that there is no difference when selecting a light or an heavy element.

To guide the eye, we have added an horizontal line to indicate the position of zero. The vertical dashed line indicates the position of the heaviest isotope used to train the network. The first estimate of the error of the NN is represented by its RMS. Assuming that the errors are normal, we set the error bar [18] to 1σ, equal to the global RMS of the model on the validation set, as done in reference [9].

As previously discussed, the simplest estimate of the error bars is obtained by taking the RMS of the NN on the validation set. The simple error bar used here can be considered probably as a very good approximation to compare with data within the range of the training, i.e. in this case for calcium isotopes with $A \leqslant 54$ and $A \leqslant 215$ for lead isotopes, while it clearly underestimates the real statistical error in the extrapolation region. The result shown in figure 4,

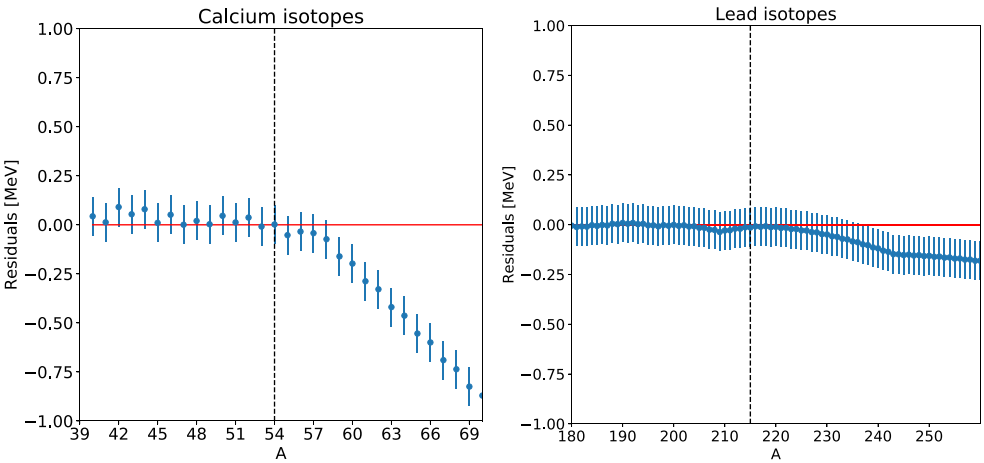

Figure 4. Evolution of the difference between the energy per particle calculated using the LD model and the NN. The vertical dashed line indicates the position of the last isotope used to train the NN. The horizontal line indicates the zero and it helps guiding the eye. See text for details.

especially in the extrapolated region, is very much dependent on several quantities as the initialisation of the weights or a different splitting of the data in training/validation. As such, the *naïve* extrapolation done here can not be considered as reliable even if accidentally some of the nuclei in the extrapolated region are still well reproduced.

A more refined error bar based on the methodologies of section 2.1 is presented in section 4.

3.2. Skyrme data

Within the NEDF theory, the total binding energy of a nucleus is written as the space integral of an energy density functional obtained using a microscopic Skyrme interaction [55, 56]. Differently from LD, the Skyrme model is non-linear in parameter space, since all the densities appearing in the various terms of the functional [57] are obtained as a self-consistent solution of the Hartree–Fock–Bogoliubov equations using an iterative procedure [58]. In the present article, we consider the data calculated in reference [48] using the SLy4 functional [47].

From the statistical point of view, the interest in using a microscopic calculation is related to the fact that the *features* are not so evident as in the case of the LD model shown in equation (3), although the overall quality in reproducing nuclear masses is similar.

Following the same procedure used for LD, we now train an NN over Skyrme data with the same strategy. In figure 5, we illustrate the evolution of the RMS as a function of the number of epochs used for the training. As discussed previously, the goal of the current paper is not to find the optimal NN, but to discuss the optimal methodology to estimate error bars.

After 15 000 epochs, we obtain an RMS of $\sigma_{\rm tr} \approx 50$ keV on the energy per particle for the training and $\sigma_{\rm val} = \approx 100$ keV for the validation set. These performances are comparable to the one of the NN trained on the LD data.

In figure 6, we show the evolution of the NN predictions along the isotopic chain of calcium and lead, in the same way as we did in figure 4. Although the drip-lines obtained using LD and SLy4 are not equal, we observe that our NN behaves quite nicely for the first isotopes beyond the last known nucleus and then it diverges.

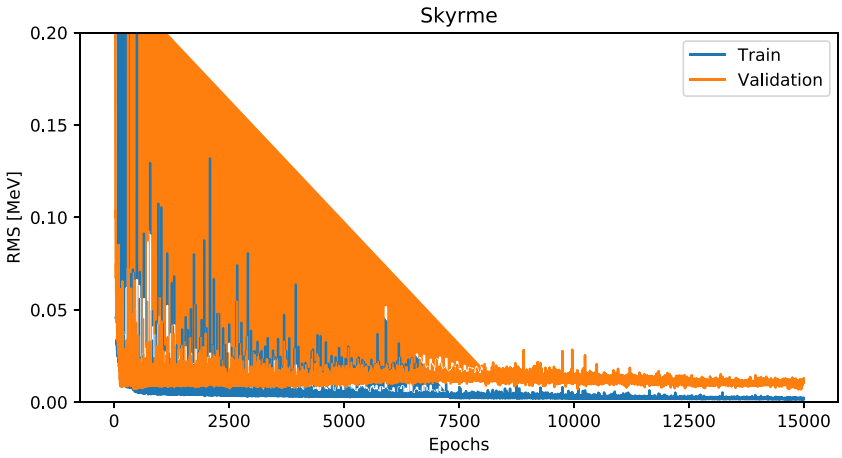

Figure 5. Same as figure 3, but using SLy4 data.

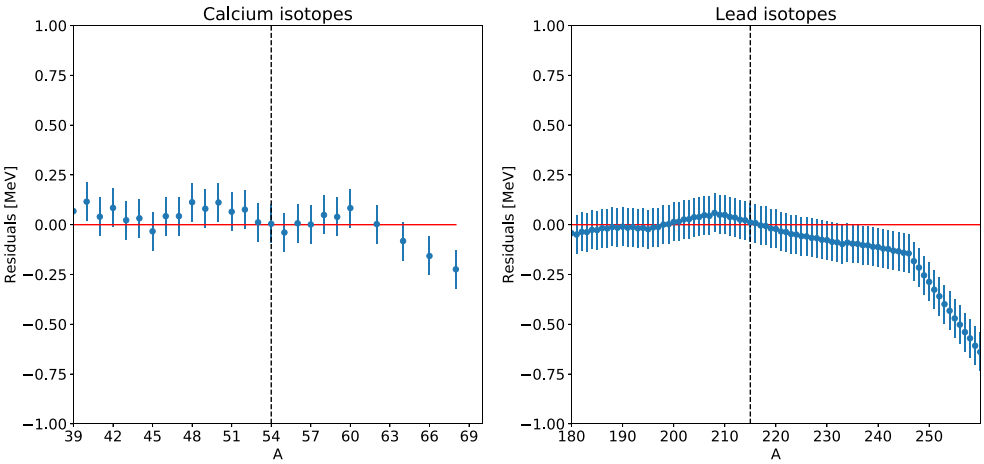

Figure 6. Same as figure 4, but for SLy4 data.

The simple error estimate based on the RMS clearly under-estimates the true error, although by chance the lead isotopes are very well reproduced by our NN in the extrapolated region.

4. Error estimate

In this section, we apply the three different methods discussed in section 2.1 to the realistic data obtained from LD and Skyrme-SLy4 models.

4.1. Epoch-averaging

By looking at figures 3 and 5, we observe that both loss functions reach a *plateau* region around 10 000 epochs. The gradient is not zero and we still observe small fluctuations. This means that

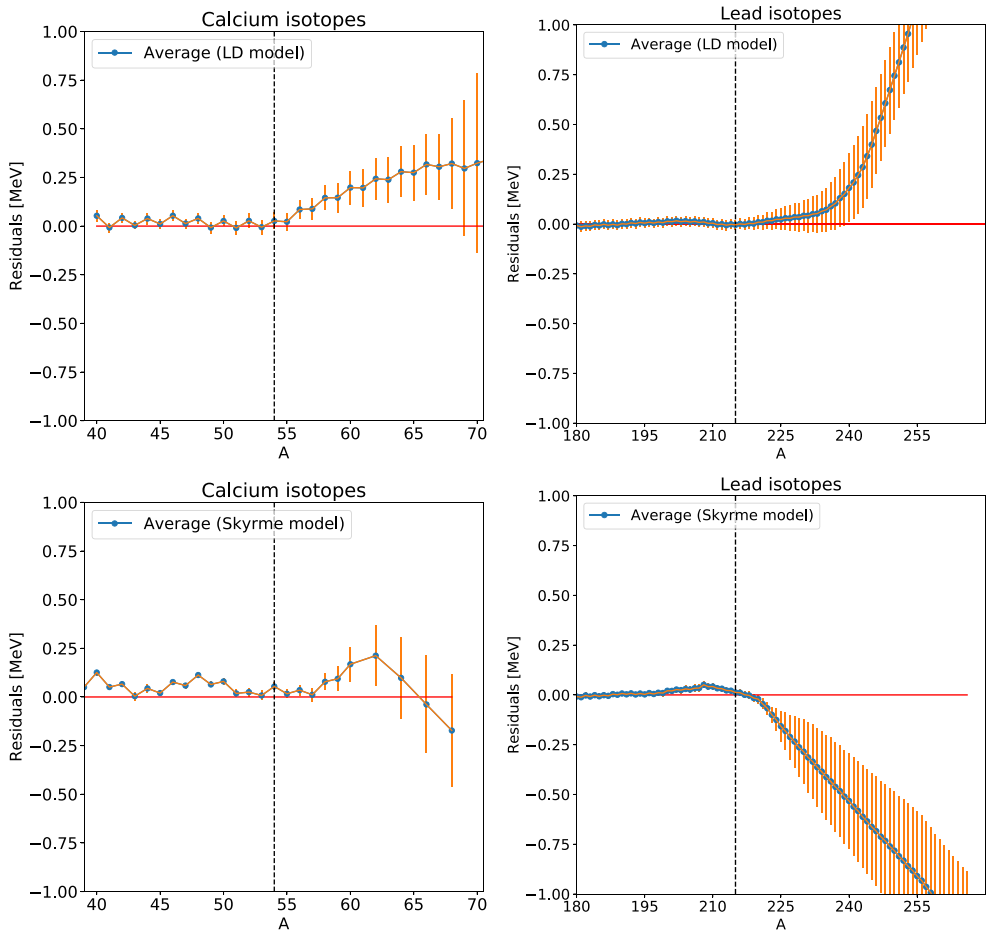

Figure 7. Same as figure 4, but using epoch-averaging to estimate error bars. On the left panels calcium isotopes and on the right panels the lead isotopes. The top row has been obtained using LD data, while the bottom row using Skyrme data. See text for details.

the weights and the biases of the NN are still fluctuating as a function of the epochs, although we expect these fluctuations to be small.

We take advantage of the epoch-averaging presented in section 2.2, by continuing the training of the network and storing the results every 1000 epochs. By doing this 100 times, we create 100 models with all slightly different parameters, but using exactly the same data set for training and validation.

We define an *average* model, by calculating the mean value of the models and the 1-σ error as the region containing 68% of the predictions. The result is presented in figure 7 for Ca and Pb isotopic chains using the synthetic data calculated using LD and Skyrme. The vertical lines indicate the position of the last experimentally known nucleus in the chain.

We observe that the error bar is very small in the region where experimental data are present (the training set). This value is even smaller than the naïve RMS shown in figures 4–6. In the region of extrapolation, i.e. $A \geqslant 54$ for calcium isotopes and $A \geqslant 215$ for lead, the error bars start to grow quite fast becoming soon way larger than the simple RMS. The predictions done

with the NN are *reasonable* for isotopes with few neutrons beyond the vertical line, but the prediction quickly deteriorates and the uncertainties becomes soon very large, and in clear disagreement with the *true* model.

Although such a procedure is quite simple and not too expensive in terms of CPU/GPU, it is worth recalling that the different networks are not independent from one another, but they are highly correlated. This can be checked by evaluating the correlation matrix between them. The consequence is that changing the training set will lead to a different prediction and different error bars, although the main outcome will remain the same.

4.2. Bootstrap

A different approach that avoids the strong dependence on the training set is based on the bootstrap method [42, 43]. In this case, we randomly split the available data into training and validation sets, reshuffling them at each bootstrap iteration. By using 100 bootstrap iterations, we have thus obtained 100 NNs, all trained for the same amount of epochs (15 000). The choice of the number of bootstrap iterations is somehow arbitrary. Thus, we have tested the evolution of the size of error bars in function of such a number. We have seen that increasing to 300 iterations the error bars do change only marginally. This is related to the structure of the data we used and it should be verified if other data are used instead. By visually inspecting the evolution of the RMS, we have checked that all networks have reached convergence with a final RMS on the training and validation sets of the same quality as the original one. As done before, we calculate the average and the variance of the various NNs to define an error bar. In this case, by examining the correlation matrix, one observes still a correlation, but much weaker than in the previous case of epoch averaging. The different NN are still partially correlated since they have been trained to strongly overlapping data sets.

In figure 8, we illustrate the evolution of the difference between the energy per particle as calculated with the LD and Skyrme models using the NN, together with the associated error bar trained using the bootstrap method.

The error bars obtained with bootstrap are robust, i.e. they do not depend on the particular choice of the initial data set, but they seem to overestimate the real precision of the NN. By moving few isotopes beyond $A = 215$ for Pb and $A = 54$ for Ca, the errors grow fast and soon reach several hundreds of keV. From the statistical point of view this error bar can be seen as a very conservative estimate of the predictive power of the model since the *true* model is always included within these error bars. This was not the case of epoch-averaging as shown in figure 7.

4.3. Dropout

We now consider the third method to estimate error bars and based on dropout [44]. As explained in detail in the simple toy-model example, during the training of the NN, we randomly turn off a given percentage of neurons for the predictions. This procedure is used in the literature to check the robustness of the network and to avoid over-fitting of the weights. As discussed in reference [44], dropout can be used as an approximation to a more involved BNN. The main advantage of dropout compared to BNNs is that the typical training time required to determine the weights is shorter. To mimic BNNs, we apply the same dropout also to the prediction. This means that every time we call the NN to obtain a value, we randomly turn off a set of neurons. This will give us the required variability to estimate an error. For simplicity (namely, writing the least amount of code), we used the option from Keras [59, 60] to use dropout at both training and prediction phases.

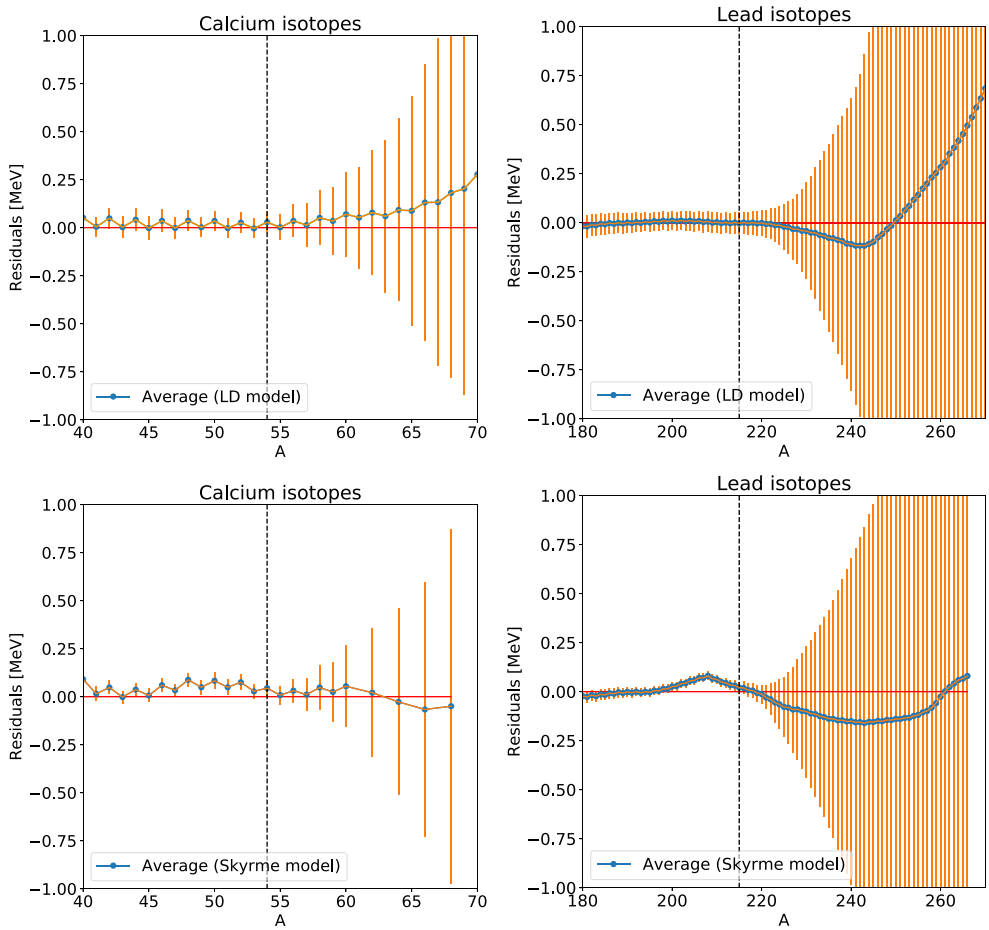

Figure 8. Same as figure 4, but using bootstrap to estimate error bars. See text for details.

The layout of the NN is the same as in previous cases, but we had to increase the number of epochs to 30 000 to let the gradient reach a stable *plateau*. The drop-out rate is arbitrary; for the present calculation we have used a rate of 5%. Other rates could be explored, but given the architecture of the NN and the relatively small number of neurons, higher dropout would lead to very poor performances. In the present case, using 5% dropout, the resulting RMS is of ≈150 keV for both the training and the validation set. This value is roughly three times worse than what we obtained using the same layout without dropout.

During the training phase, some neurons are silenced, forcing the other neurons to learn to compensate the missing ones. By spreading the information learned by one neuron to another, the dropout effectively improves the performance by assembling the predictions of different models trained in parallel. By default, the dropout is not used during the predictions, and the output of all neurons is considered. In this way, the prediction process is deterministic and reproducible.

If, however, the dropout is used at prediction time, every time a prediction is launched a different result is possible. This leads to a distribution function for each and every prediction. If we assume a normal distribution, by averaging and taking the standard deviation, we obtain

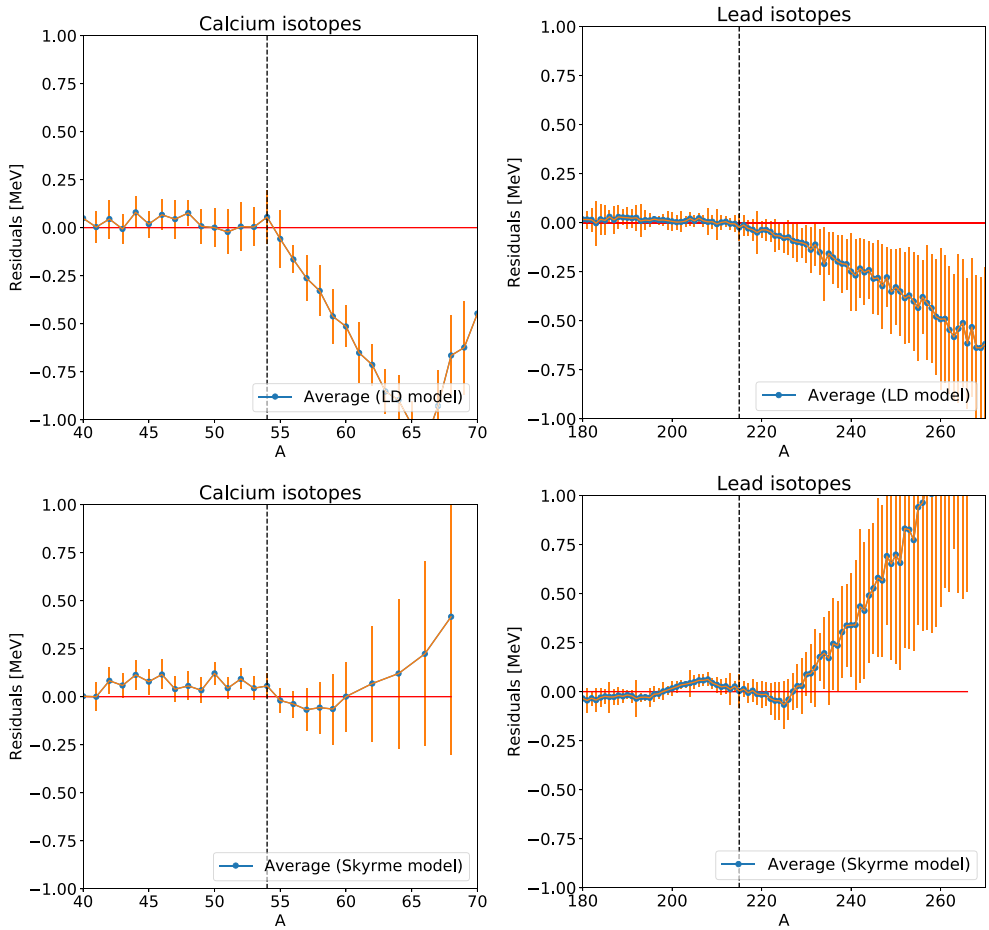

Figure 9. Same as figure 4, but using dropout to estimate error bars. See text for details.

an estimation of the bar error. Of course, the normal distribution is not strictly required, but in order to assess the RMS, the second moment should be finite.

With a finite amount of neurons, the possible outcomes are limited: for an NN with ten neurons and a drop out rate of 10%, every time a prediction is made one of the neurons is silenced. This leads to only ten possible distinct values for the prediction.

The result is reported in figure 9 for the two data-sets used and for two isotopic chains. The quality of the predictions in the region used for the training is slightly worse, but in the extrapolated region we observe that the behaviour of the resulting error bar is somehow intermediate between the epoch-averaging and the bootstrap.

Similarly to the bootstrap case, the use of dropout reduces the dependence of the outcome on the specific choice of training/validation set, but using much less CPU time. The error bars contain the *true* model only for a few isotopes beyond the last known nucleus in the LD model, while for the Skyrme model the error bars do a better job. This may be accidental and not easy to predict without knowing the *true* model.

5. Conclusions

We have presented three different methodologies to estimate the error bars of the prediction obtained with an NN. Through a simple toy-example, we have illustrated in detail the calculations of error bars using epoch-averaging, bootstrapping and dropout. We have applied these three methods to the more realistic case of nuclear binding energies. To benchmark the accuracy of the error bars and predictions, we have used synthetic data obtained from two well known models: the LD and the Skyrme SLy4. By observing the quality of the predictions and the structure of the error bars on two representative isotopic chains, we conclude that bootstrap and dropout are robust methods since they do not depend too much on the particular choice of the initial training set as in the case of epoch-averaging. The bootstrap tends to provide the largest error bars that contain the *true* value of a large set of nuclei, and unfortunately these error bars are so large that the prediction itself lacks any relevance. The dropout seems more promising in providing a more reasonable error bar. Moreover, it has an actual regularisation effect during the training phase thus reducing the difference between the performance on the training and the validation set. Using dropout, we obtain a distribution for the predictions. This allows an *a posteriori* analysis of the results, and thus an error estimation. This mimics the behaviour of a more complex BNN with a reduced computational cost. The optimal dropout rate for training and prediction has not been explored in detail. Some additional programming effort focusing on the dependence of error bars, and more generally on the extrapolation performance, is required.

Other parameters may also increase the variability of our predictions, as for example the initialisation of the weights. Different choices of weights may lead to slightly different results and they should also be taken into account for a more detailed analysis.

We consider that these results are a first step towards a machine learning model for predicting the nuclear masses based on the available, experimental masses. The key ingredients for these models are robust error estimations and validation schemes as discussed in this paper.

Another ingredient that is required for good performance and for generalisation beyond the known masses is a solid feature engineering approach. We illustrated with the toy model how the usage of the right feature can dramatically improve the predictions. While the example may seem artificial, it serves the purpose to express the need for carefully designed features. This means that NN cannot replace modelling, but only complement it.

Acknowledgments

The authors thank R Lasseri for useful discussions in the initial phase of this work and D Regnier for useful comments. This work has been supported by STFC Grant No. ST/P003885/1.

Data availability statement

All data that support the findings of this study are included within the article (and any supplementary files).

ORCID iDs

A Pastore https://orcid.org/0000-0003-3354-6432

References

[1] Friedman J, Hastie T and Tibshirani R 2001 *The Elements of Statistical Learning* (*Springer Series in Statistics*) (New York: Springer)
[2] Clark J W 1999 *Scientific Applications of Neural Nets* ed J W Clark, T Lindenau and M L Ristig (Berlin: Springer) pp 1–96
[3] Gernoth K A 1999 *Scientific Applications of Neural Nets* ed J W Clark, T Lindenau and M L Ristig (Berlin: Springer) pp 139–69
[4] Athanassopoulos S, Mavrommatis E, Gernoth K A and Clark J W 2004 *Nucl. Phys.* A **743** 222
[5] Athanassopoulos S, Mavrommatis E, Gernoth K and Clark J W 2004 *Nucl. Phys.* A **743** 222–35
[6] Utama R, Piekarewicz J and Prosper H 2016 *Phys. Rev.* C **93** 014311
[7] Utama R and Piekarewicz J 2017 *Phys. Rev.* C **96** 044308
[8] Neufcourt L, Cao Y, Nazarewicz W and Viens F 2018 *Phys. Rev.* C **98** 034318
[9] Pastore A, Neill D, Powell H, Medler K and Barton C 2020 *Phys. Rev.* C **101** 035804
[10] Akkoyun S, Bayram T, Kara S O and Sinan A 2013 *J. Phys. G: Nucl. Part. Phys.* **40** 055106
[11] Lasseri R-D, Regnier D, Ebran J-P and Penon A 2020 *Phys. Rev. Lett.* **124** 162502
[12] Cybenko G 1989 *Math. Control Signal Syst.* **2** 303
[13] Hornik K 1991 *Neural Netw.* **4** 251
[14] Kortelainen M, Lesinski T, Moré J, Nazarewicz W, Sarich J, Schunck N, Stoitsov M and Wild S 2010 *Phys. Rev.* C **82** 024313
[15] Kortelainen M *et al* 2014 *Phys. Rev.* C **89** 054314
[16] Dobaczewski J, Nazarewicz W and Reinhard P-G 2014 *J. Phys. G: Nucl. Part. Phys.* **41** 074001
[17] Roca-Maza X, Paar N and Colò G 2015 *J. Phys. G: Nucl. Part. Phys.* **42** 034033
[18] Barlow R 1989 *The Manchester Physics Series* (New York: Wiley)
[19] Gernoth K A and Clark J W 1995 *Comput. Phys. Commun.* **88** 1
[20] Weizsäcker C F v 1935 *Z. Phys.* **96** 431
[21] Bender M, Heenen P-H and Reinhard P-G 2003 *Rev. Mod. Phys.* **75** 121
[22] Wang M, Audi G, Kondev F G, Huang W J, Naimi S and Xu X 2017 *Chin. Phys.* C **41** 030003
[23] Xu K, Li J, Zhang M, Du S S, Kawarabayashi K-i and Jegelka S 2020 arXiv:2009.11848
[24] Leshno M, Lin V Y, Pinkus A and Schocken S 1993 *Neural Netw.* **6** 861
[25] Ruder S 2016 arXiv:abs/1609.04747
[26] Mishkin D and Matas J 2015 arXiv:abs/1511.06422
[27] Carnini M and Pastore A 2020 *J. Phys. G: Nucl. Part. Phys.* **47** 082001
[28] Karlik B and Olgac A V 2011 *Int. J. Artif. Intell. Expert Syst.* **1** 111
[29] Lu Z, Pu H, Wang F, Hu Z and Wang L 2017 arXiv:1709.02540
[30] Glorot X and Bengio Y 2010 *Proc. of the 13th Int. Conf. on Artificial Intelligence and Statistics* pp 249–56
[31] Yam J Y F and Chow T W S 2000 *Neurocomputing* **30** 219
[32] Kingma D P and Ba J 2014 arXiv:1412.6980
[33] Heaton J 2017 arXiv:1701.07852
[34] Brownlee J 2020 *Machine Learning Mastery with Python* https://machinelearningmastery.com/machine-learning-with-python/
[35] Shelley M, Becker P, Gration A and Pastore A 2019 *Acta Phys. Pol.* B **12** 649
[36] Neal R M 2012 *Bayesian Learning for Neural Networks* vol 118 (New York: Springer)
[37] Lampinen J and Vehtari A 2001 *Neural Netw.* **14** 257
[38] Brownlee J 2018 *Better Deep Learning* https://machinelearningmastery.com/better-deep-learning/
[39] Polyak B 1990 *Avtomatica i Telemekhanika* **7** 98
[40] Polyak B T and Juditsky A B 1992 *SIAM J. Control Optim.* **30** 838
[41] Ruppert D 1988 *Technical Report* Cornell University
[42] Efron B and Tibshirani R 1986 *Stat. Sci.* **1** 54–75
[43] Pastore A 2019 *J. Phys. G: Nucl. Part. Phys.* **46** 052001
[44] Srivastava N, Hinton G, Krizhevsky A, Sutskever I and Salakhutdinov R 2014 *J. Mach. Learn. Res.* **15** 1929–58
[45] Cowan J J, Thielemann F-K and Truran J W 1991 *Phys. Rep.* **208** 267
[46] Chamel N and Haensel P 2008 *Living Rev. Relativ.* **11** 10
[47] Chabanat E, Bonche P, Haensel P, Meyer J and Schaeffer R 1998 *Nucl. Phys.* A **635** 231
[48] Stoitsov M V, Dobaczewski J, Nazarewicz W and Borycki P 2006 *Int. J. Mass Spectrom.* **251** 243
[49] Goriely S, Hilaire S, Girod M and Péru S 2009 *Phys. Rev. Lett.* **102** 242501

[50] Duflo J and Zuker A P 1995 *Phys. Rev.* C **52** R23
[51] Moeller P and Nix R 1994 *Technical Report* Los Alamos National Laboratory
[52] Sobiczewski A and Litvinov Y A 2014 *Phys. Rev.* C **90** 017302
[53] Wang M, Audi G, Wapstra A H, Kondev F G, MacCormick M, Xu X and Pfeiffer B 2012 *Chin. Phys.* C **36** 1603
[54] Anil M U, Malik T and Banerjee K 2020 arXiv:2004.14196
[55] Perlińska E, Rohoziński S, Dobaczewski J and Nazarewicz W 2004 *Phys. Rev.* C **69** 014316
[56] Becker P, Davesne D, Meyer J, Navarro J and Pastore A 2017 *Phys. Rev.* C **96** 044330
[57] Tondeur F 1983 *Phys. Lett.* B **123** 139
[58] Ring P and Schuck P 2004 *The Nuclear Many-Body Problem* (Springer)
[59] Chollet F 2017 *Deep Learning with Python* 1st edn (New York: Manning Publications)
[60] Chollet F *et al* 2015 *Keras* https://github.com/fchollet/keras