Solving the Schrödinger equation with machine learning

Author: Javier Rozalén Sarmiento
Facultat de Física, Universitat de Barcelona, Diagonal 645, 08028 Barcelona, Spain.

Advisor: Arnau Rios Huguet

Abstract: The Schrödinger equation is known to be a great tool for understanding small physical systems, yet an analytical solution exists solely for some simple cases. As a consequence of this, a wide variety of numerical methods are used to solve real systems, and machine learning methods, namely Artificial Neural Networks (ANNs), are amidst the newest. In this work we use an ANN to find the ground state wave function of the deuteron -this is, the nuclear two-body bound state-, achieving energy values that are within 0.05% of the exact results and wave functions that overlap up to 99.9997% with the exact ones. We also compare the performance of a single-layer ANN against a two-layer ANN, the latter not showing significant improvements over the former.

I. INTRODUCTION

Artificial Neural Networks are being used more and more in science as an alternative method to solve computationally expensive problems [1]. Fields like quantum chemistry are currently seeing significant improvements both in terms of efficiency and accuracy due to this methodology. Amongst the best examples of this is FermiNet [2], an ANN that uses a variational *ab-initio* method to solve the many-electron problem, obtaining better results than the variational methods typically used to approach such problems. In this work we approach the deuteron in a similar way, albeit much simpler in spirit.

The methodology that we use to solve the deuteron problem, as concerns the design and implementation of the ANN, stems from Ref. [3], published in 2020 with my advisor as the co-author. In this paper, the ground state wave function of the deuteron in momentum space is found using a single-layer ANN that acts as the trial wave function in a variational method. The main aim of this work is to reproduce the results obtained in that paper, and also to extend the neural network by adding an extra hidden layer and compare the performance of both the original and the extended ANN.

This document is structured in the following manner: in section II we briefly go over the structure of a simple ANN; in section III we tackle the deuteron problem and explain the computational implementation. In the next section we comment on our extension of the network in Ref. [3] and in section V we show some numerical results obtained with both the initial ANN and the extended ANN.

II. NEURAL NETWORKS

From the mathematical point of view, an ANN is a function that depends on a given number of parameters and a special kind of function. A single-layer ANN with one input and one output reads:

$$f(x) = \sum_{i=1}^{N_{\text{hid}}} W_i^{(2)} \sigma\left(W_i^{(1)} x + B_i\right). \tag{1}$$

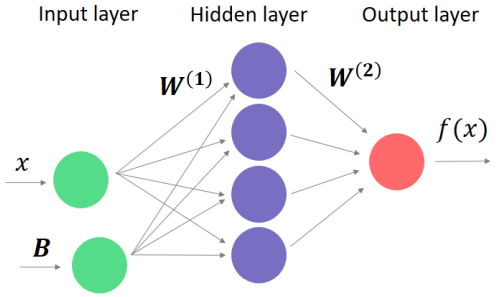

FIG. 1: Architecture of a one-layer, one-output ANN with bias $\mathbf{B}$ in the hidden layer and four hidden neurons.

Here, the parameters $W_i^{(1)}$, $W_i^{(2)}$, B_i, $i = 1, \ldots, N$ are the components of the vectors $\mathbf{W^{(1)}}$, $\mathbf{W^{(2)}}$, $\mathbf{B}$, and σ is the activation function, i.e., a continuous, non-decreasing function such that $\lim_{x\to-\infty} \sigma = 0$ and $\lim_{x\to\infty} \sigma = 1$. A theorem exists asserting that, if N is large enough, a function like (1) can uniformly approximate any continuous function [4]; this is usually referred to as *Universal Approximation Theorem*. Therefore we must find the vectors $\mathbf{W^{(1)}}$,$\mathbf{W^{(2)}}$,$\mathbf{B}$ that best approximate the desired function.

In order to understand the process of finding these vectors, the usual graph representation comes in handy; the graph corresponding to Eq. (1) would be the one in Fig. 1.

Computationally, the ANN learns a function in a finite number of points, the *training set*, and if properly trained it is able to transfer that knowledge to other points as well (*testing set*). One iteration of the learning process is as follows: every input x_i of the training set is passed to the input layer, from where it is "propagated forward" to each neuron of the next layer, and so on. In this propagation each neuron uses a weight parameter $W_j^{(i)}$ to perform an affine transformation on the neuron input (see Eq. (1)).

The final output $f(x)$ is then used to compute the cost function, which is a measure of the quality of the prediction and has a global minimum where the prediction

is exact. Finally, this cost function is in turn passed to a minimisation algorithm, usually called *optimizer*, which "backpropagates" the error to each neuron, i.e., it changes the weights of all the neurons in a way that will let the next prediction be more accurate than the previous one. The amount by which the parameters are varied is typically proportional to a (hyperparameter) constant called *learning rate*, which is to be chosen carefully. This process is repeated until the ANN converges to the desired function.

III. COMPUTATIONAL SET-UP

We use an *ab-initio* variational method: starting from an arbitrary wave function that depends on N parameters, we aim to find the parameters that minimize the total energy. The ANN plays the role of the variational wave function, and the parameters of the network are the parameters of the wave function. From here it follows naturally that the energy is the cost function, which is precisely the function that is minimised throughout the training process; thus, we have direct access to the ground state wave function. Defining $\mathcal{W} = \{\mathbf{W^{(1)}}, \mathbf{W^{(2)}}, \mathbf{B}\}$, the energy in terms of the ANN parameters reads:

$$E^{\mathcal{W}} = \frac{\langle \psi^{\mathcal{W}}_{\text{ANN}} | \hat{H} | \psi^{\mathcal{W}}_{\text{ANN}} \rangle}{\langle \psi^{\mathcal{W}}_{\text{ANN}} | \psi^{\mathcal{W}}_{\text{ANN}} \rangle}. \tag{2}$$

We solve the problem in a 64-point, one-dimensional lattice that we use as the momentum space (training set), with q' (relative momentum) initially in the interval $[0, 1]$. As concerns the distribution of q', we initially take $q'_i = x_i$, where x_i are the x-points used in the Gauss-Legendre integration method with $N = 64$ points. This allows us to use the same points to compute both the wave function and the necessary quadratures via Gauss-Legendre. In order to capture the high momentum components of the potential energy we perform the following auxiliary transformation over the initial training set:

$$q_i = \frac{q_{\text{max}}}{\tan \frac{\pi}{2} q'_N} \tan \frac{\pi}{2} q'_i \,, \tag{3}$$

where q'_N is the biggest q value of the initial mesh, and $q_{\text{max}} = 500\,\text{fm}^{-1}$. This extends the mesh up to $500\,\text{fm}^{-1}$ whilst providing us with a dense mesh at low momenta.

We know beforehand that the only possible states for the deuteron are the ones with angular momenta $L = 0, L = 2$, and this information allows us to choose a proper ANN architecture. In fact, we use the one shown in Fig. 1, but now with two output neurons instead of one: one for the $|\psi^{L=0}(q)\rangle$ state (or $S-$state) and one for the $|\psi^{L=2}(q)\rangle$ state (or $D-$state). The number of parameters in terms of the number of hidden neurons is now $N = 4N_{\text{hid}}$.

Following [3], we take three steps to train the network. First, we initialize the parameters to uniform random values: $\mathbf{W^{(1)}} \in [-1, 0)$, $\mathbf{W^{(2)}} \in [0, 1)$ and $\mathbf{B} \in [-1, 1)$, and we choose a *sigmoid* as the activation function. This step is important, for the ANN is completely random at first, and by setting sensible initial values we avoid early large values for the wave function.

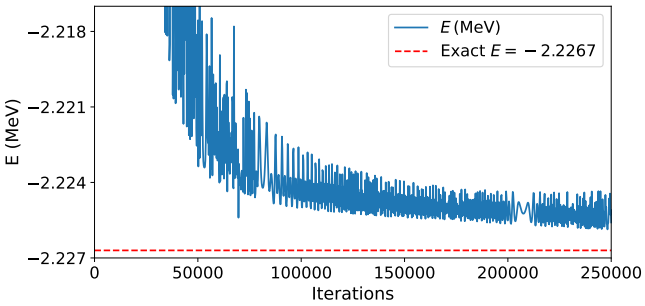

FIG. 2: Total energy as a function of the number of iterations.

In the second step we train the ANN to mimic a target function that bears a certain degree of physical meaning and is somewhat similar to the exact one: $\psi^L_{\text{targ}}(q) = q^L e^{-\frac{\xi^2 q^2}{2}}$, with $\xi = 1.5\,\text{fm}$ to match expected results. We use the overlap

$$K^L = \frac{\langle \psi^L_{\text{targ}} | \psi^L_{\text{ANN}} \rangle^2}{\langle \psi^L_{\text{targ}} | \psi^L_{\text{targ}} \rangle \langle \psi^L_{\text{ANN}} | \psi^L_{\text{ANN}} \rangle} \tag{4}$$

to compute the cost function, defined as:

$$C = (K^S - 1)^2 + (K^D - 1)^2 \,. \tag{5}$$

Eq. (5) is zero if, and only if, $K^S = K^D = 1$; thus, in order to minimize the cost function, both the S-state and the D-state must converge to the target functions. The choice of optimizer here is RMSProp ([1], p.18), which is fast and dynamically varies the learning rate to adapt locally to the cost function. Training a medium-sized ANN ($N_{\text{hid}} \approx 20$) usually takes about 10^4 iterations; despite that, this step is unnecessary in some configurations.

In the third and final step we train the pre-trained model from step two to find the actual ground state wave function just by setting the total energy (2) as the new cost function. The strong interaction of choice is N3LO Entem-Machleidt nucleon-nucleon force [5], which we can directly embed in our code since it is a momentum-dependent potential. In Fig. 2 we can observe the evolution of the energy (blue) as the number of iterations increases. The energy displays a clear tendency to decrease, and it does so more slowly as training time increases, finally reaching a value of $E^{\mathcal{W}} \approx -2.225\,\text{MeV}$. This corresponds to a model with $N_{\text{hid}} = 20$ hidden neurons and a learning rate of 10^{-2}.

Fig. 3 displays the wave function obtained at the end of the training process of the same model: on the right panel we can observe the D-state corresponding to the ANN (solid blue); the wave function obtained via exact diagonalization (dashed red) is shown as well as a reference. On the left panel we have the same information

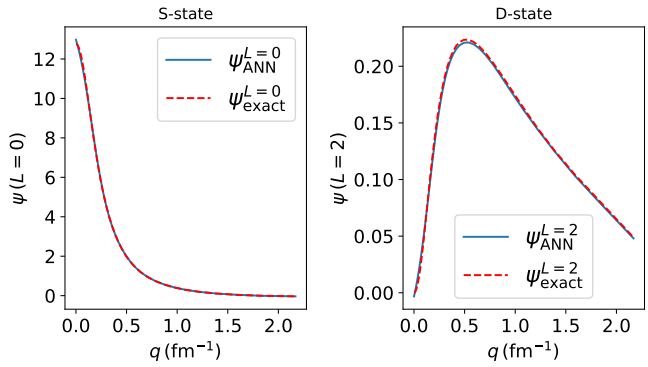

FIG. 3: Wave function computed by the ANN (blue) compared to the wave function obtained via exact diagonalisation (red).

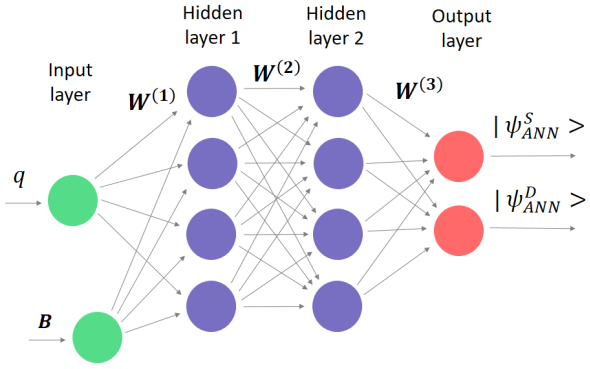

FIG. 4: Architecture of a two-layer ANN with two outputs and a single input with bias in the first hidden layer. Both hidden layers have the same number of neurons.

for the S-state. Both states match almost perfectly the expected results, overlapping with the exact function by $K^S \approx K^D \approx 0.99995$.

To implement this method we have used Python as the programming language in tandem with the PyTorch library, which contains pre-made subroutines that allow for a more direct implementation of all ANN-related operations. The full version of the code can be found in the GitHub repository: https://github.com/javier-rozalen/deuteron.git

IV. EXTENDING THE ANN

We have just seen that a one-layer ANN can already yield decent results. Despite that, it is widely known that deep neural networks, i.e. multi-layer ANNs, often perform notably better than one-layer ANNs. Therefore we have extended the ANN we have been using hitherto in this paper by adding an extra hidden layer, without modifying anything else.

The architecture of this ANN is depicted in Fig. 4, and the function form reads:

$$\psi^L_{\text{ANN}}(q) = \sum_{i=N_{\text{hid}}/2+1}^{N_{\text{hid}}} W^{(3)}_{i,L}\sigma\left(\sum_{j=1}^{N_{\text{hid}}/2} W^{(2)}_{j,L}\sigma\left(W^{(1)}_{j,L}q + B_{j,L}\right)\right). \tag{6}$$

The number of parameters in terms of the number of hidden neurons is now $N = N_{\text{hid}}\left(2 + \frac{1}{4}N_{\text{hid}}\right)$. Multi-layer networks rest under the umbrella of the Universal Approximation Theorem as well; in fact, the theorem in Ref. [4] is a particular case of a bigger theorem that includes any kind of ANNs [6].

The first thing that we notice when we train a network with this architecture is that it is almost indispensable to start the training from a pre-trained model, as opposed to the one-layer case. In addition, we even find more difficulty in obtaining good pre-trained models. This is partially due to the fact that we use the same intervals for the initial parameters $\mathcal{W}$ as for the one-layer case, and a quick look at Eq. (1) and Eq. (6) confirms this: for a given number of hidden neurons N_{hid} there are more terms in the sum in the two-layer case than in the one-layer case, which entails larger early values of the wave function and therefore possible divergencies. More specific results obtained with this network are discussed in section V.

V. NUMERICAL RESULTS

A. Discussion of the results

The quantities that we use as indicators of the quality of our models are the energy, the fidelity (as in Eq. (4)) and the probability of the D-state.

In Fig. 5 we present the results obtained after testing both the one-layer and two-layer networks with a sigmoid, RMSProp as the optimizer and a learning rate of 10^{-2} for different numbers of N_{hid}. As for the two-layer ANN there exist better hyperparameters, but we place more importance in isolating architecture effects to understand them rather than in obtaining accurate results; using different hyperparameters for the two networks would thereby hinder our analysis.

The solid black lines in Fig. 5 indicate the central values of each quantity, whereas the shaded areas show the errors associated to the central values. The nature of the errors will be discussed in subsection B. We observe a clear tendency: the more hidden neurons, the better the results. In fact, we can observe lower energies and higher fidelities at high N_{hid}, which was somewhat expected; the energy increase at $N_{\text{hid}} = 80$ for 2 layers is an isolated anomaly, as further tests have evinced). For 1 layer, increasing N_{hid} from 20 to 100 translates into a 0.02% decrease in energy ($\approx 0.04\,\text{keV}$), a $7\cdot10^{-4}$% increase in K^S and a $4\cdot10^{-4}$% increase in K^D, whilst for 2 layers these

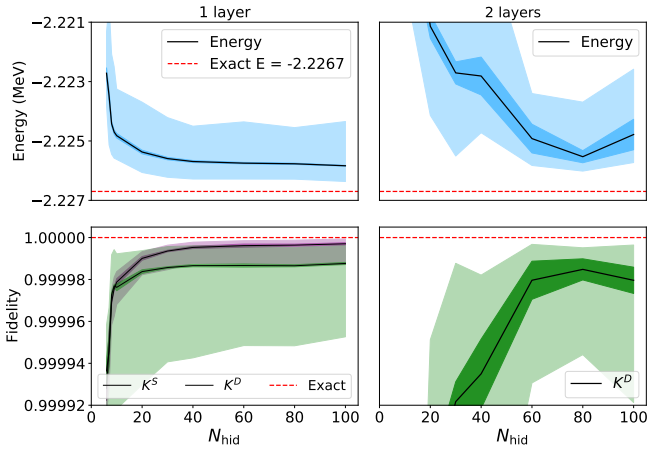

FIG. 5: Left panels: energy (top) and fidelity (bottom) as a function of the number of hidden neurons for the one-layer ANN. Right panels: the same for the two-layer ANN, omitting the D-state fidelity for clarity.

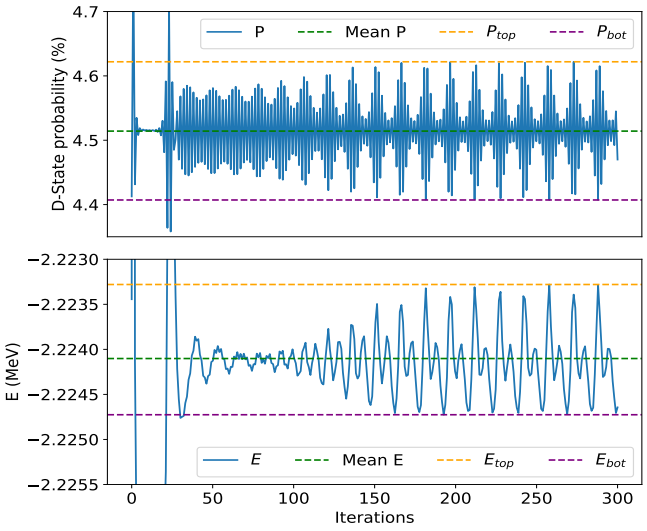

FIG. 6: Top: D-state probability post-evolution of a trained model with a single-layer architecture and $N_{\text{hid}} = 40$ hidden neurons. Bottom: total energy evolution of the same model as in the top panel.

percentages are 0.2% ($\approx 4.4\,\text{keV}$), 0.006% and 0.01%. The best absolute results are: $E^{\mathcal{W}} = -2.2258\,\text{MeV}$, $K^S = 99,9997\%$, $K^D = 99,999\%$ (1 layer) and $E^{\mathcal{W}} = -2.2255\,\text{MeV}$, $K^S = 99,999\%$, $K^D = 99,9998\%$ (2 layers), the exact values being $E = -2.2267\,\text{MeV}$ (obtained via exact diagonalization), $K^S = K^D = 100\%$.

Fig. 5 is to be interpreted carefully, nonetheless. Firstly, each architecture has been trained during the same number of iterations, and this is the main reason behind the form of the energy plots: we have found that, for $N_{\text{hid}} \gtrsim 10$, ANNs with a lower number of neurons do not learn worse, only slower. Indeed, if we take a model with low N_{hid} and we train it for long enough, the energy reaches similar values to those obtained for high N_{hid}. This does not happen for $N_{\text{hid}} \lesssim 10$, however: in these cases the ANN has too few parameters to adjust and the result becomes independent of the number of iterations at a certain point.

There is another factor to take into consideration, and it is that with the one-layer ANN, for high values of N_{hid} it is increasingly difficult to find adequate initial parameters, i.e., ones that avoid early divergences and vanishing gradients. For example, when $N_{\text{hid}} = 10$, it takes ~ 10 different initializations to obtain non-divergent models, whereas for $N_{\text{hid}} = 100$ it takes ~ 30 different initializations to obtain 10 good models. We believe that this is due to the fact that we use random initial values within the same intervals as in section III; if we look at Eq. (1) we realize that, in the extreme case of $N_{\text{hid}} = 100$ we are adding 100 terms together, and if the initial parameters $\mathbf{W^{(2)}}$ happen to be large enough this could lead to an early divergence. On the other hand, if $\mathbf{W^{(1)}}$ and $\mathbf{B}$ make the argument of the sigmoid too positive or too negative, the gradient of the sigmoid approaches zero and the change of parameters is insignificant, leading to what is known as *frozen neurons*: neurons that no longer contribute to the learning process.

The two-layer ANN undergoes a similar phenomenon but in reverse: it is more difficult to train networks with little neurons. One possible explanation for this is that, since we are using a sigmoid as our activation function, vanishing gradient becomes more problematic as we now have two hidden layers, and this leads to frozen neurons. In addition to this, having few neurons increases the probability that all of them become frozen. This is the reason why we believe that, for multi-layer networks it is better to use an activation function like *softplus*, which has an infinite range and a larger gradient.

B. Error analysis

The shaded areas of Fig. 5 represent two different types of errors: stochastic (dark) and oscillating (softer hue) (only the former is considered in Ref. [3]). To understand the necessity for the two kinds of errors we have to understand the relation between the training process and the final trained ANN. When training the neural network, as the energy approaches the global minimum, oscillations appear (see Fig. 2). This is not surprising, for we are seeking the minimum value of a function by taking finite steps in the direction where the gradient is negative. We are interested in finding the energy of the ground state, and if we suppose that the oscillations happen around the minimum value of $E^{\mathcal{W}}$ we can estimate this value by averaging these oscillations; therefore, the oscillations provide us with a measure of the likelihood that the minimum value of $E^{\mathcal{W}}$ lies within the range of the amplitude, and are hence an error source.

In order to determine this error we take a trained model and we let it evolve for $\sim$ 300 iterations; this

number is small enough so that the mean value will not change during this process, and also big enough so that periodicity in the oscillations is observed.

We then leverage the last 200 iterations to calculate the mean value of the energy and also to determine the maximum and minimum values, which will be taken as the top and bottom error associated with the mean value. This is illustrated in the bottom panel of Fig. 6: the energy (blue) oscillates around the mean value (green dashed line), and the top and bottom bounds of this oscillations are shown in yellow and purple dashed lines. In this particular example, the mean value of the energy is $E^{\mathcal{W}} = -2.2241\,\text{MeV}$, with oscillation errors of $\delta E^{\mathcal{W}}_{+} = 0.8\,\text{keV}$, $\delta E^{\mathcal{W}}_{-} = 0.6\,\text{keV}$; these are the typical values obtained for almost any number of neurons.

On the top panel of Fig. 6 we show a similar analysis for the D-state probability, with values: $P^D = 4.51\%$, $\delta P^D_{+} = \delta P^D_{-} = 0.1\%$; in fact, the same analysis is done for the overlap, which is shown in Fig. 5 (and not shown in Fig. 6 for brevity).

In order to take into account the stochastic factor we repeat the process above 10 times for each network configuration (N_{hid}), and we extract the central value of the energy by taking the average of the 10 mean values. The final oscillation errors also include this stochasticity: both top and bottom bounds are calculated by averaging the 10 invididual bounds. This method suggests that, given a cost function with no multiple minimums of similar heights (hence all oscillations happen around the same minimum) the mean values of $E^{\mathcal{W}}$ should all be similar. This, in tandem with the fact that we estimate the stochastic $\delta E^{\mathcal{W}}$ as the standard deviation, is the reason why in Fig. 5 we observe small stochastic errors and comparatively bigger oscillation errors.

VI. CONCLUSIONS AND FUTURE OUTLOOK

Firstly, we have been able to successfully reproduce most of the work done in [3], following roughly the same methodology; this comprises sections II and III. The results obtained are in perfect concordance with the ones in Ref. [3]. We have also provided a new source of error for the energy, the oscillating error (discussed in section V, subsection B).

Besides this, we have extended the reference ANN by adding a hidden layer, and it has proven to perform nearly as accurately as the original ANN with the particular hyperparameters of our choice. The comparison between both networks has provided us with a deeper insight into the intricate mechanism of ANNs (see discussion in section V, subsection A).

We have explored a possible extension of this work consisting in approaching the deuteron problem with deeper ANNs (more than 2 layers). According to the tests carried out, the need for an activation function like softplus gains more importance as the number of layers increases; otherwise, the gradients rapidly vanish and the ANN learns no more. The results obtained are not any better than the ones obtained with the simpler ANNs used hitherto in this work. We believe that this is due to the simplicity of our approach; for instance, if we were to design our trial wave function as a Slater determinant of monoparticular wave functions, as done in [2], we would then probably benefit from the extra layers. This kind of approaches might yield better results and are worth exploring, although they are beyond the aim of this work.

VII. ACKNOWLEDGEMENTS

I want to thank first and foremost my advisor Dr. Arnau Rios Huguet, who has more than willingly helped me throughout the whole process, and also Professor Bruno Juliá Díaz for his valuable comments. I want to thank my family and my partner as well for all the support that I have received from them.

[1] P. Mehta, M. Bukov, C.H. Wang, et al. A high-bias, low-variance introduction to machine learning for physicists. *Phys. Reports*, 810, 2019.

[2] D. Pfau, J.S. Spencer, A.G. de G. Matthews, et al. Ab-initio solution of the many-electron Schrödinger equation with deep neural networks. *Phys. Rev. Research*, 2, 2020.

[3] J.W.T. Keeble and A. Rios. Machine learning the deuteron. *Phys. Lett. B*, 809, 2020.

[4] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. of Control, Signals, and Systems (MCSS)*, 2(4), 1989.

[5] D. R. Entem and R. Machleidt. Accurate charge-dependent nucleon-nucleon potential at fourth order of chiral perturbation theory. *Phys. Rev. C*, 68, 2003.

[6] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2), 1991.

[7] A. Gunes, B.A. Pearlmutter, A. Andreyevich, et al. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153), 2018.

[8] Pytorch tutorials. `https://pytorch.org/tutorials/beginner/basics/intro.html`. Accessed: 2021-04-15.

[9] G. Carleo and M. Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325), 2017.

[10] H. Saito. Method to solve quantum few-body problems with artificial neural networks. *Journal of the Phys. Society of Japan*, 87(7), 2018.