Machine learning solutions for the two-dimensional quantum harmonic oscillator

Author: León Begiristain Ribó
Facultat de Física, Universitat de Barcelona, Diagonal 645, 08028 Barcelona, Spain.

Advisor: Arnau Rios Huguet

Abstract: In this work, I have used Artificial Neural Networks to find the ground state of the 2D quantum harmonic oscillator. I have trained networks in two different ways: by using a mesh of points and by using Monte Carlo methods. I have used the analytical solution of the problem to benchmark the quality of the results of both methods, obtaining overlaps up to 0.99998 in the case of the mesh training and 0.9989 in the case of Monte Carlo. The relative errors in the energy are 0.03% and 1.1% respectively. I have shown the effects of the number of neurons and the learning rate on the overall performance of the network. Training with Monte Carlo shows faster convergence, while training on the mesh gets closer to the exact energy.

I. INTRODUCTION

Artificial intelligence is a rapidly-evolving field that has demonstrated a huge potential. In the last few years, uncountable applications have emerged, not only in computer science but also extending to many other fields of science. Particularly in physics, many works have shown the usefulness of machine learning techniques, and the use of Artificial Neural Networks (ANNs) is growing rapidly. For instance, in quantum mechanics, Ref. [1] solved few-body bosons systems and Ref. [2] solved spin many-body systems. Since then, many other works have used ANNs to solve different quantum systems [3, 4]. Yet, there have not been many applications in multidimensional systems.

In this work, I have focused on the quantum harmonic oscillator in 2D. I have chosen this system because it has a simple analytical solution, and therefore, is adequate to begin solving multidimensional problems. The exact solution of the problem is useful to benchmark the quality of the results given by the neural network.

I have found the ground state of the system using two different training methodologies. The aim has been to study the performance of both techniques, showing the effect of variations of key parameters on the training. To accomplish this, I provide an introductory explanation about neural networks, and I take a look at the computational set-up. Finally, I present the numerical results, comparing the performance of both methods.

II. ARTIFICIAL NEURAL NETWORKS

ANNs are computational systems that approximate functions using some parameters, usually known as weights. They are organized in layers: there is always an input and an output layer, and there can be also a number of hidden layers. Mathematically, there are weights between each layer, that are applied to the previous layer as a linear application. In order to allow non-linear behavior, an activation function is applied between each layer. Typically, an activation function σ is a non-decreasing function that usually goes from $\sigma(x = -\infty) = 0$ to $\sigma(x = \infty) = 1$. The ANN architecture I have used in this work is shown in Fig. 1. It has one hidden layer of N_{hid} neurons, and can mathematically be written as

$$\varphi^{\mathcal{W}}(\vec{r}) = \sum_{i=1}^{N_{hid}} W_i^{(2)} \sigma \left(\sum_{j=1,2} W_{ij}^{(1)} r_j + b_i \right), \qquad (1)$$

where $\mathcal{W} = \{W^{(1)}, W^{(2)}, b\}$ are the weights and $\vec{r} = (r_1, r_2) = (x, y)$, the spatial coordinates. The parameters that control the training, such as the number of neurons and the learning rate, are called hyperparameters and are explained later in this work.

In order to approximate the desired function, appropriate values for all the weights must be found. The ANN starts with random initial values, which are progressively improved by little updates at each step. To achieve this, an iterative process is followed, usually known as the "training" of the neural network. The iterations of the training are called epochs.

The way of testing the quality of some given weights is by using a cost function. The cost function takes an ANN state and returns a scalar, the loss value. The training minimizes the loss, and therefore, the cost function works as a way of measuring the distance to the target. For this work, I have used the energy as the cost function.

At each epoch, the loss of the current ANN state is computed and given to the optimizer, which updates the weights. The amount by which parameters are changed is determined by a hyperparameter called learning rate, lr. Once the weights of the network are updated, the cost function is computed once again using the ANN, and the whole process is repeated. The training of the network can be stopped when the loss converges, but in this work I have trained the networks for a fixed number of epochs.

With this procedure and using a neural network with enough neurons, the wave function can be obtained. In fact, the *Universal approximation theorem* ensures that it is possible to approximate any continuous function by superposition of activation functions [5].

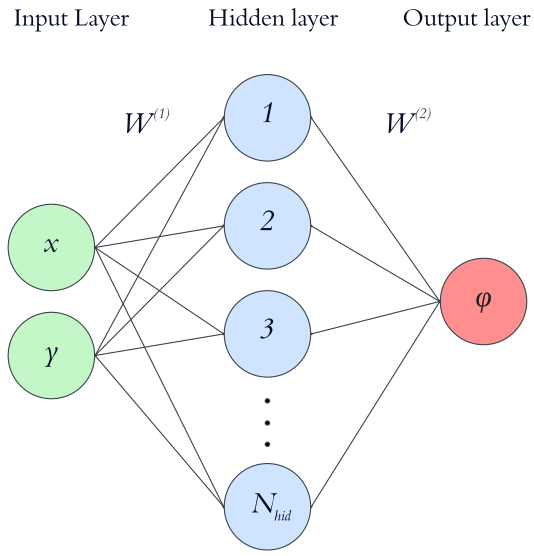

FIG. 1: Architecture of the neural network: two inputs, the spatial coordinates x and y (green); a number N_{hid} of hidden neurons (blue) and a single output, the wave function (red). The total number of weights is $4N_{hid}$: $2N_{hid}$ between the input and the hidden layer, N_{hid} in the bias and N_{hid} between the hidden layer and the output. I do not show bias weights for simplicity.

III. THE QUANTUM HARMONIC OSCILLATOR

The Hamiltonian of the two-dimensional quantum harmonic oscillator is

$$\hat{\mathcal{H}} = -\frac{\hbar^2}{2m}(\partial_x^2 + \partial_y^2) + \frac{1}{2}m(\omega_x^2 x^2 + \omega_y^2 y^2), \tag{2}$$

where m is the mass of the particle, x and y are the cartesian coordinates and ω_x and ω_y are the angular frequencies. Introducing harmonic oscillator units of length and energy, $x_{ho} = \sqrt{\hbar/m\omega_x}$ and $E_{ho} = \hbar\omega_x$, the expressions are simplified. The solution of the Schrödinger equation is

$$\varphi_{n_x,n_y}(x,y) = \frac{2^{n_x+n_y} n_x! n_y!}{\sqrt{\pi}} e^{-\frac{1}{2}(x^2+\frac{\omega_y}{\omega_x}y^2)} H_{n_x}(x)\, H_{n_y}\left(\sqrt{\frac{\omega_y}{\omega_x}}y\right), \tag{3}$$

where n_x and n_y are the quantum numbers and $H_{n_x}(x)$ and $H_{n_y}(\sqrt{\omega_y/\omega_x}y)$ are the Hermite polynomials. Using these units, the Hamiltonian becomes

$$\hat{\mathcal{H}} = -\frac{1}{2}(\partial_x^2 + \partial_y^2) + \frac{1}{2}\left(x^2 + \left(\frac{\omega_y}{\omega_x}\right)^2 y^2\right), \tag{4}$$

and the ground state of the wave function ($n_x = n_y = 0$) is

$$\varphi_{0,0}(x) = \frac{1}{2\pi}\sqrt{\frac{\omega_y}{\omega_x}} e^{-\frac{1}{2}(x^2+\omega_y/\omega_x y^2)}. \tag{5}$$

In these units, the ground state has an energy $E_0 = \frac{1}{2}(1+\omega_y/\omega_x)$, depending on the ratio between the angular frequencies.

IV. COMPUTATIONAL SET-UP

A. Architecture and training cycle

The architecture of the ANN that I have used is shown in Fig. 1. I have also tested networks with two hidden layers, which perform much worse, as shown in Ref. [3].

I have used the network to find the ground state of the oscillator, using the energy as the cost function. The energy is computed as the normalized expectation value of the Hamiltonian,

$$E^{\mathcal{W}} = \frac{1}{2}\frac{\left\langle \varphi^{\mathcal{W}} \middle| -(\partial_x^2+\partial_y^2) + x^2 + \frac{\omega_y}{\omega_y}y^2 \middle| \varphi^{\mathcal{W}} \right\rangle}{\langle \varphi^{\mathcal{W}} \,|\, \varphi^{\mathcal{W}} \rangle}. \tag{6}$$

In the kinetic energy term, I have avoided computing the second derivative of the wave function integrating by parts. The remaining term only contains first derivatives. The python framework, PyTorch [6], used in this work offers a way to compute this derivative automatically.

Once the energy is obtained, the gradients with respect to the weights are computed by backpropagation. After that, I use the Adam optimizer [7] to calculate the new weights. In a few words, Adam takes the learning rate α, the gradient of the weights $\nabla_W \mathcal{L}$ and other constants $(\beta_1, \beta_2, \epsilon)$, and uses them to update the weights,

$$\mathcal{W}_{i+1} = \mathcal{W}_i - \alpha f(\nabla_{\mathcal{W}} E^{\mathcal{W}}, \beta_1, \beta_2, \epsilon), \tag{7}$$

where $f(\nabla_{\mathcal{W}} E, \beta_1, \beta_2, \epsilon)$ is a function of these parameters. The training continues at the next epoch by computing the wave function with the new values of the ANN.

B. Training with fixed points

I now present the first training method I have used in this work. It consists in using points in a square mesh that does not change during the whole training. For this work, I have chosen the trapezoidal rule to compute the energy of the wave functions on the mesh. The 2D trapezoidal integral can be expressed as a sum over all the points multiplied by a weighting factor,

$$\int d\vec{r}\varphi^*(\vec{r})\hat{\mathcal{H}}\varphi(\vec{r}) = h^2 \sum_{i,j} w_{ij}\varphi^*(x_i,y_j)\hat{\mathcal{H}}\varphi(x_i,y_j), \tag{8}$$

where h is the lateral size of the squares in the mesh, and $w_{ij} = 1$ everywhere except in the borders of the mesh. Nevertheless, knowing that the wave function tends to zero at infinity and selecting a big enough mesh, I have taken $w_{ij} = 1$ everywhere, neglecting the effect of the borders.

Although this method is computationally efficient, it presents two problems. First, as the network just learns at the points of the mesh, the wave function may have the wrong shape outside or between the points of the grid. Over-fitting [8] refers to the phenomenon of a network finding a solution to the problem that only works with the training set. The second problem is that the number of points in the input scales with the number of dimensions d, as L^d, where L is the number of points on each edge of the mesh. Going to higher dimensions, it is unfeasible to continue using this method. I now discuss a second training method that can overcome these limitations.

C. Training with Monte Carlo

The second way of training the ANN is using random point distributions that stochastically evolve using the Metropolis Hastings (MH) algorithm [9]. I have updated the point distribution every 10 epochs, with a proposal distribution width $\sigma_{MH} = 1$.

I have estimated the wave function integrals using Monte Carlo techniques, a method known as QMC [10]. First, I compute the local energy of each point,

$$\epsilon(x,y) = -\frac{1}{2}\frac{(\partial_x^2+\partial_y^2)\varphi(x,y)}{\varphi(x,y)} + \frac{1}{2}\left(x^2 + \left(\frac{\omega_y}{\omega_x}\right)^2 y^2\right). \tag{9}$$

Then, I compute the effective number of samplers as the expectation value of the ratio between the probability distribution of the new wave function, $|\varphi^{\mathcal{W}}(\vec{r})|^2$, and the distribution of the previous MH step $p(\vec{r})$,

$$N = \mathop{\mathrm{E}}_{\vec{r}\sim p}\left(\frac{|\varphi^{\mathcal{W}}(\vec{r})|^2}{p(\vec{r})}\right), \tag{10}$$

where the $\vec{r}$ points are distributed following $p(\vec{r})$. $p(\vec{r})$ is the Born probability associated to $\varphi^{\mathcal{W}}$ at the epoch when the MH step is performed. Finally, I obtain the total energy as the expectation value of the local energies multiplied by the same factor,

$$E^{\mathcal{W}} = \frac{1}{N}\mathop{\mathrm{E}}_{\vec{r}\sim p}\left(\epsilon(\vec{r})\frac{|\varphi^{\mathcal{W}}(\vec{r})|^2}{p(\vec{r})}\right), \tag{11}$$

This factor corrects for the difference between both distributions, as $p(\vec{r})$ is not updated at every epoch. Using this approach is known as importance sampling.

The variance of the energy is computed as

$$\sigma^2 = \mathop{\mathrm{E}}_{\vec{r}\sim p}\left(\epsilon(\vec{r}) - E^{\mathcal{W}})^2\frac{|\varphi^{\mathcal{W}}(\vec{r})|^2}{p(\vec{r})}\right), \tag{12}$$

and I have used it to estimate the uncertainty in the integral as $\delta E^{\mathcal{W}} \approx \sigma/\sqrt{n_s}$.

The advantage of the MC method is that, as the training set is continuously evolving, the network is forced to learn the shape of the whole wave function, not on a mesh. This way, I expect over-fitting to be avoided.

Nevertheless, there is a limitation. Once in a while, random points will be sampled far from the rest of the distribution, and thus, away from the region where the network is mainly trained. It is likely that the local energies of these points are estimated incorrectly. To avoid this, I have applied two preventive measures. First, clipping the energy. This means I have limited the local energy of the points to a certain deviation around the mean. In particular, I have clipped the energy to eight times the l_1-*norm* of the local energy. Secondly, an envelope has been applied to the wave function. The envelope ensures the wave function gets to zero far from the origin, helping with normalization and reducing the problem with distant points. Computationally, it means multiplying the wave function with a function that goes to zero at infinity. For this work, I have chosen a Gaussian envelope with an amplitude $\lambda = 2$,

$$\varphi^{\mathcal{W}}_{MC}(x,y) = \varphi^{\mathcal{W}}_{ANN}(x,y)e^{-\frac{1}{2\lambda^2}(x^2+y^2)}. \tag{13}$$

While introducing some bias, these two measures have been really useful in order to ensure a consistent performance of the network. Both measures are explained more in depth in Ref. [11].

V. RESULTS

I now turn to discussing the results. I have looked for optimum values for the number of neurons in the hidden layer, N_{hid}, and the learning rate, lr. Fig. 2 shows the results for the training in the mesh, and Fig. 3 for the training with MC. For the first I have used a mesh with 50×50 points in $x, y \in [-5, 5]$, and for the second, $n_s = 1000$ sampling points. Using the training with MC, more hyperparameter must be adjusted, for example, the frequency of distribution updates. If the distribution is not updated frequently, the network will adapt to those points. This causes jumps in the energy when $p(\vec{r})$ is updated, as it is likely that the network will not have the correct shape in the new points. On the other hand, updating the distribution often causes a noisier, but more consistent evolution.

Looking at Fig. 2, I find similar initial energies for all values of N_{hid} and lr. Since the initial values for the weights are selected randomly, some variation through different runs is expected. Nevertheless, as the initial weights are distributed uniformly, the resulting wave functions are mostly constant. Knowing this, the observed energy $E \approx 9$ is expected, as one can estimate the energy of a constant wave function on a 50×50 mesh around $E \approx 8.7$.

There is also a clear distinction between the initial energies of both training methods. This is caused by the envelope used in the MC training. It makes the initial wave function more similar to a Gaussian, resulting in a lower energy.

Looking at the evolution with different N_{hid}, I find that with few neurons the wave function can only adopt

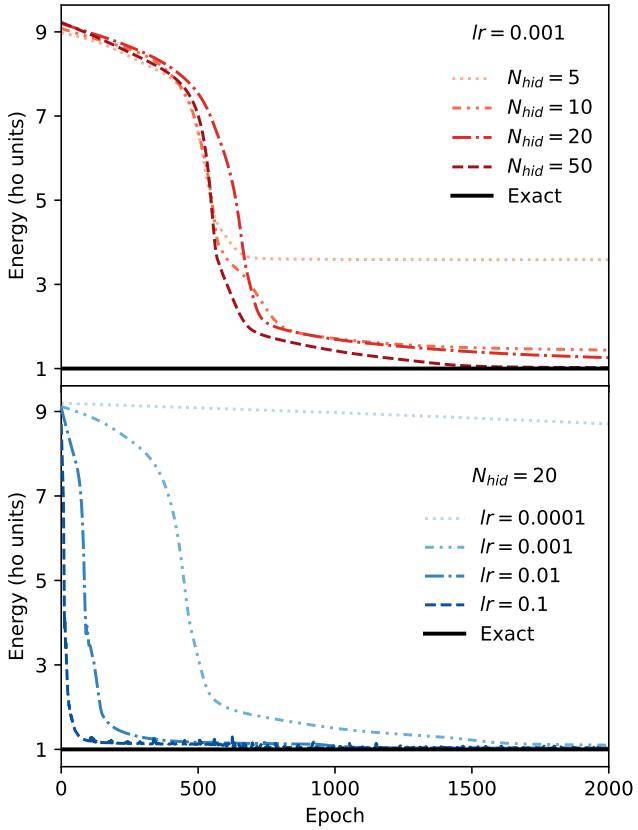

FIG. 2: Energy evolution during training in a fixed mesh of 50×50 points for the isotropic oscillator ($\omega_x = \omega_y$). In the top panel, the evolution for different numbers of neurons is shown for a learning rate of $lr = 0.001$. In the bottom panel, the effect of changes in the learning rate for $N_{hid} = 20$ neurons. In both panels, the energy of the exact solution is shown as a horizontal solid line (black).

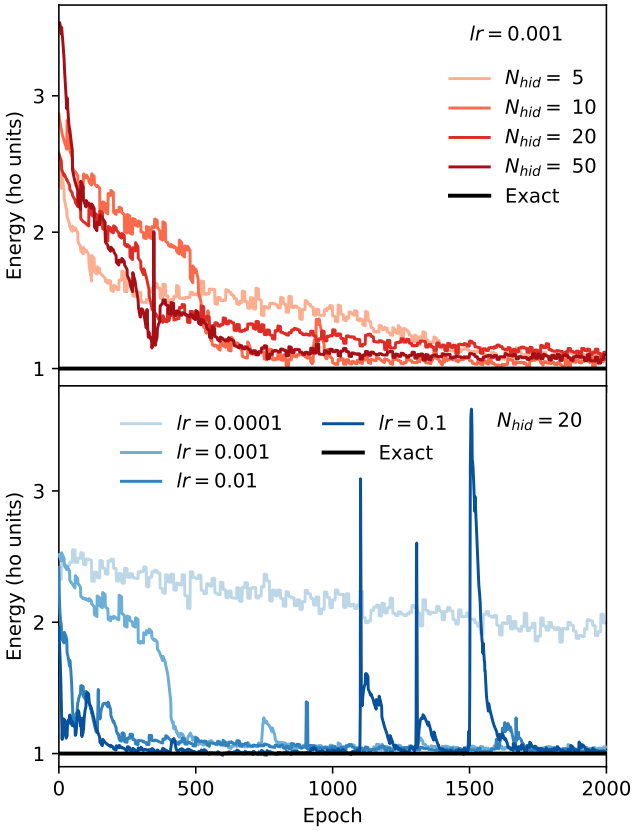

FIG. 3: The same as Fig. 2 but for networks trained with the Monte Carlo method. All networks have been trained using $n_s = 1000$ sampling points and the point distribution has been updated every 10 epochs. The uncertainty associated to the Monte Carlo integral is not shown for clarity.

simple shapes. This means that, if the target function is too complex, there are simply not enough neurons to adopt its shape. For example, in the mesh training with $N_{hid} = 5$, the energy flattens far from the exact energy. Meanwhile, a big network is easier to adjust, as many combinations of the parameter can result in the same output. Training in the mesh, the largest N_{hid} has presented the fastest convergence. Nevertheless, the network can be too complicated, resulting in over-fitting in the case of fixed points. I will show later that the networks I have used have not been over-fitted.

Comparing both panels, I find that N_{hid} has a bigger effect in the training with the fixed mesh. Training with MC, all the networks get to relatively small values in the energy, under 5% relative error, in the first two thousand epochs. On the other hand, small networks are not able to get that close when training with the mesh points.

The learning rate hyperparameter also changes the training of the network. Small values result in a slow optimization, as can be seen for the $lr = 0.0001$ in both trainings. On the contrary, training with high learning rates is more erratic and not so consistent. The network with $lr = 0.1$ in the MC training is the fastest to get under the 1% error, but presents an unstable convergence with high spikes, and is therefore not a good value for this training method.

In general, both training methods show similar responses to variations on the learning rate. However, there is a noticeable difference when going to high values of lr. When using $lr = 0.1$ in the MC training, the energy is unstable, while the same value shows consistent results when training on the mesh.

Finally, I have compared the wave functions at the end of the training to the exact solution, both by plotting the probability distributions (Fig. 4) and by computing the overlap between wave functions. For this test, networks have been trained to solve an anisotropic oscillator with relative frequencies $\omega_y/\omega_x = 1/2$. I have computed the overlaps using the trapezoidal rule, evaluating the wave functions on a dense mesh of 1000×1000 points. This way, the possible over-fitting when using the fixed mesh training could be noticed. For the training on the mesh, the overlap between wave functions is $\langle \varphi^{\mathcal{W}}_{mesh} \,|\, \varphi_{exact} \rangle = 0.99998$, and the relative error

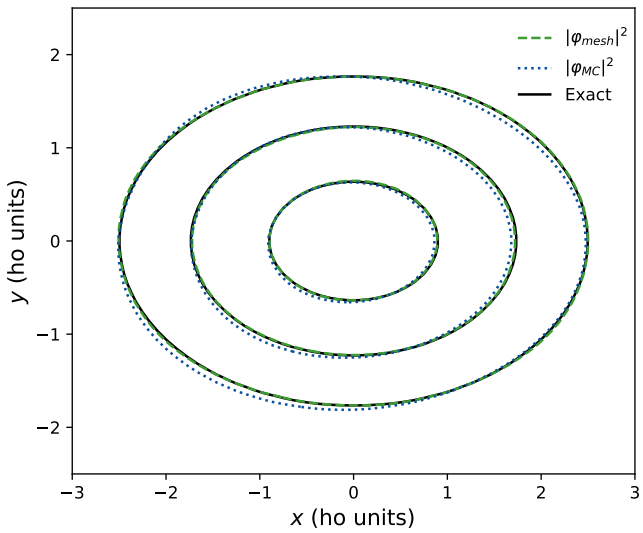

FIG. 4: Contours at 0.05, 0.1 and 0.15 of the probability distribution of the wave function for an oscillator with relative frequencies $\omega_y/\omega_x = 1/2$. The training with a mesh is shown in dashed lines (green) and the training with MC in dotted lines (blue), both obtained with $lr = 0.001$ and $N_{hid} = 20$. The exact solution is also shown as a solid line (black).

between energies, $\delta E^{\mathcal{W}}_{mesh} = \Delta E_{mesh}/E_{exact} = 0.03\%$. Therefore, it can be stated that the network has not been over-fitted. On the other hand, the results for the Monte Carlo training are $\langle \varphi^{\mathcal{W}}_{MC} \,|\, \varphi_{exact} \rangle = 0.9989$ and $\delta E_{MC} = 1.1\%$.

VI. CONCLUSIONS

In this work, I have found the ground state of the two-dimensional harmonic oscillator using ANNs with two different training methodologies. I have shown that both training methods are capable of finding the correct shape of the target wave function. The training with a fixed mesh has shown to be able of getting closer to the analytic result, with a relative error in energy of $\delta E_{mesh} = 0.03\%$, compared to the $\delta E_{MC} = 1.1\%$ using the Monte Carlo.

Training with random points has shown a faster convergence of the energy, as they rapidly get close to the analytical results. Nevertheless, the training with the mesh points has gotten closer to the exact energy, and it has been shown that there has not been over-fitting. Therefore, it can be concluded that while slower, the fixed mesh ensures a more consistent and stable solution of the problem in two-dimensions.

This work can be continued in two different ways. First, increasing the number of dimensions, where the MC training should be more effective. Secondly, using the network to find excited states of the oscillator by looking for wave functions that are orthogonal to the ground state.

The code of the whole work is available at: https://github.com/leonbegiristain/ML-QHO.

Acknowledgments

I want to express gratitude to my advisor, Dr. Arnau Rios, for all his help during these months. I also want to thank Javier Rozalén, Amir Azzam and James Keeble for their useful comments during our meetings and my friends, family and partner for their support during the whole work.

[1] H. Saito. Method to solve quantum few-body problems with artificial neural networks. *Journal of the Physical Society of Japan*, 87(7):074002, 2018.
[2] G. Carleo et al. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, 2017.
[3] J. Rozalén et al. Machine learning the deuteron: new architectures and uncertainty quantification. *arXiv:2205.12795*, 2022.
[4] JWT Keeble et al. Machine learning one-dimensional spinless trapped fermionic systems with neural-network quantum states. *arXiv:2304.04725*, 2023.
[5] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
[6] A. Paszke et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
[7] DP. Kingma et al. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.
[8] X. Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.
[9] S. Brooks et al. *Handbook of markov chain monte carlo*. CRC press, 2011.
[10] F. Becca et al. *Quantum Monte Carlo approaches for correlated systems*. Cambridge University Press, 2017.
[11] D. Pfau et al. Ab initio solution of the many-electron Schrödinger equation with deep neural networks. *Physical Review Research*, 2(3):033429, 2020.