Institut de Ciències del Cosmos
UNIVERSITAT DE BARCELONA

EXCELENCIA MARÍA DE MAEZTU 2020-2023

Transfer Learning For Many-Body Systems

Amir Azzam

Supervised by: A. Rios & J. Rozalén

10^{th} of July, 2022

Abstract: We study the ground-state properties of fully-polarized, trapped, one-dimensional fermions interacting through a Gaussian potential using a variational wavefunction represented by an antisymmetric artificial neural network. We optimize the network parameters by minimizing the energy of a four-particle system with machine learning techniques. We introduce several methods to enhance the efficiency, accuracy, and scalability of the artificial neural network approach. We use Early Stopping to terminate the training when the energy converges. We apply Transfer Learning to pre-train the network with different interactions or particle numbers to reduce the memory cost and improve the performance. We employ a re-weighting scheme to speed up the Metropolis-Hastings sampling for estimating the energy expectation value. We show that, in particular, this re-weighting scheme can speed up the training time by up to a factor of 10.

Keywords: Transfer Learning, Artificial Neural Networks, Neural-Network Quantum State, Early Stopping, Metropolis-Hastings, Monte-Carlo

Amir Azzam: aazzamaz41@alumnes.url.edu, ameerazzam2001@hotmail.com

Acknowledgments

I am extremely grateful to Dr. Arnau Rios and Javier Rozalén Sarmiento for their help and supervision during this project. I would also like to thank Dr. James Keeble for his support throughout the project.

Contents

1 Introduction

The past decade has seen a tremendous rise in machine learning techniques applied to different industries, including healthcare, finance, natural language processing, and more. Some of these techniques have been proven beneficial to simulate physical systems in many different ways (1; 2; 3). One of the most notable benefits is finding the minimum energy of quantum many-body systems, which can be done using Neural-Network quantum states (NQSs)(3). These networks were first introduced back in 2017 by Carleo et al. (1). They were one of the first to train a network to find the ground state energy of complex interacting quantum systems. Another remarkable uses of these NQSs came back in 2020 when the researchers at *DeepMind* introduced for the first time FermiNet (4). In their work, they introduced a neural network that is able to represent a fermionic wavefunction ansatz for many-electron systems. These networks managed to predict the dissociation curves of the nitrogen molecule and hydrogen chain more accurately than the state-of-the-art quantum chemistry approaches at the time. This shows that NQSs can potentially speed up the performance of the many-body simulations (5).

NQSs are a variational representation of the many-body wavefunction in terms of artificial neural networks (ANNs) (1). These networks solve the Schrödinger equation and find the system's ground state energy by optimizing the network's parameters. Moreover, NQSs can incorporate the symmetries of the Hamiltonian into the network, which allows the simulation of not only ground states but also excited states (6).

One example of such a complex system is the one-dimensional fermionic system studied in Ref. (7) by Keeble et al. This paper investigated the case of one-dimensional spinless fermions confined in a harmonic trap with finite-range Gaussian interactions. The authors employed NQSs to solve the Hamiltonian of this system for up to $A = 6$ particles. In this project, we build upon their work and examine various techniques that can enhance the performance of their NQS. Some of these techniques aim to increase the generalization and convergence time of the network, while others focus on reducing the training time and resource consumption of the network.

We will explore three main techniques to enhance the performance and efficiency of the network. First, Early Stopping is used as a regularization method to monitor the convergence of the network and prevent overfitting (8). Second, Transfer Learning is applied to leverage the knowledge gained from previous tasks and improve the generalization and resource consumption of the network (9). Two transfer learning scenarios are investigated: transferring across different interaction strengths and transferring across different numbers of particles. Third, a scheduled re-weighting algorithm is used to accelerate the network training time by adjusting the Metropolis-Hasting sampling frequency used in this project (10).

This report will be divided into four sections. First, in Sec. 2 we will go through the project's fundamentals and methodologies. Then, Sec. 3 will explain the methods used to implement the various techniques, as well as the benefits each technique brings to the model. After that, Sec. 4 will explore some of the findings of this study, as well as the effects of combining these techniques to enhance overall network performance. Finally, Sec. 5 will discuss the project's conclusions as well as future possible directions of work.

2 Methodology

2.1 Formulation of the problem Hamiltonian

This project focuses on a system of A identical, fully-polarized fermions with mass m trapped in a harmonic potential with frequency ω. Due to the fully-polarized nature of the fermions, their spin is disregarded. The fermions interact through a finite-range Gaussian inter-particle potential, resulting in the following Hamiltonian,

$$\hat{H} = \sum_{i=1}^{A} \left[-\frac{\hbar^2}{2m}\nabla_i^2 + \frac{1}{2}m\omega^2 x_i^2 \right] + \frac{V}{\sqrt{2\pi}\sigma} \sum_{i<j}^{A} e^{-\frac{(x_i - x_j)^2}{2\sigma^2}}, \tag{1}$$

where x_i represents the position of the i-th particle, V denotes the strength of the Gaussian interaction, and σ denotes its range.

The choice of the Gaussian interaction is made for practical reasons. First, the ultimate goal of this project is to be applied to nuclear physics and these types of finite range interactions are quite relevant (11). Second, although contact interactions do not contribute to the energy of the system, Gaussian interactions allow us to reach the limit of contact interactions as the interaction range $\sigma \to 0$. Finally, due to the simplicity of the associated integrals, these types of interactions can be easily handled by many-body simulations, including methods like exact diagonalization (12) or stochastic methods (13). Some of these methods are limited by the number of particles they can simulate, but they are very useful for benchmarking the results of this project.

Throughout this project, harmonic oscillator (HO) units are used. As a result, we can re-write the Hamiltonian in Eq. (1) as,

$$\hat{H} = \sum_{i=1}^{A} \left[-\frac{1}{2}\nabla_i^2 + \frac{1}{2}x_i^2 \right] + \frac{V_0}{\sqrt{2\pi}\sigma_0} \sum_{i<j}^{A} e^{-\frac{(x_i - x_j)^2}{2\sigma_0^2}}, \tag{2}$$

where the length scale is defined as $a_{ho} = \sqrt{\frac{\hbar}{m\omega}}$ and the energies are measured in units of $\hbar\omega$. The interaction is redefined so that $\sigma_0 = \sigma/a_{ho}$ and $V_0 = V/(a_{ho}\hbar\omega)$. It is important to note that as the fermions are spinless, the many-body wavefunction $\Psi(x_1, ..., x_A)$ is anti-symmetric with respect to the position of the particles x. Consequentially, the wavefunction cancels whenever $x_i = x_j$ for two particles $i \neq j$, and the contact interactions do not contribute to the energy of the system.

2.2 Artificial Neural Networks

This project employs a particular kind of ANNs to model the wavefunction. To provide the necessary background, we first introduce the concept and mechanisms of ANNs and then describe the specific variant that we use in this work. ANNs are computing systems that simulate the structure and function of biological neural networks in the brain and nervous system (14). ANNs are composed of layers of artificial neurons that communicate with each other through weighted connections. Each artificial neuron receives a signal from the previous layer, processes it with some non-linear function, and sends it to the next layer. The first layer of an ANN is the input layer, which receives the input features (i.e. input data) encoded in a suitable format. The last layer of an ANN is the output layer, which produces the output according to the desired task (15).

Additionally, a multi-layer ANN has one or more hidden layers between the input and the output layers. These layers can vary in shape and connectivity, depending on

the architecture of the network (16). ANNs can be trained using different approaches, depending on whether the data is labeled or not. Labeled data means that each input has a corresponding output or result. Unlabeled data means that there is no predefined output or result for each input.

On the one hand, we have supervised learning (17) which is a machine learning approach that uses labeled data to train an ANN to predict outcomes for new data. The ANN learns from the input-output pairs and adjusts its weights and biases based on some learning rule and uses an error function that measures the difference between the actual output and the desired output. Supervised learning is mostly used in classification or regression problems (18).

On the other hand, we have unsupervised learning (19) which is a machine learning approach that uses unlabeled data to train an ANN to discover hidden patterns or structures in the data without human intervention. The ANN does not have a predefined output or result to compare with, so it learns by finding similarities or differences among the inputs. This is done by defining a loss function that measures the performance of the network, then the network is trained to optimize this loss. Unsupervised learning is most known for its use in clustering, association, or dimensionality reduction problems.

In this project, we will use unsupervised learning to train our network. This will be done by minimizing a loss function that depends on the energy associated with the wavefunction represented by the network. For more details about ANNs and the different components involved in training a neural network, the reader can refer to Appendix B.

2.3 Neural-Network Quantum States

NQSs are a class of variational quantum states. These states are represented by an ANN and are parameterized using the network weights, thus allowing the state to change and take form as the network trains. The network's cost function is typically the expectation value of the Hamiltonian,

$$E = \frac{\langle \Psi_\theta | \hat{H} | \Psi_\theta \rangle}{\langle \Psi_\theta | \Psi_\theta \rangle} = \frac{\int d\mathbf{x} \Psi_\theta^\dagger(\mathbf{x}) \hat{H} \Psi_\theta(\mathbf{x})}{\int d\mathbf{x} |\Psi_\theta(\mathbf{x})|^2}, \tag{3}$$

which needs to be minimized given the current NQS. Here Ψ_θ is the wavefunction of the system represented by the NQS and parameterized by the variational parameters θ, and $\mathbf{x} = (x_1, x_2, ..., x_A)$ are the positions of the A particles in the system. Then, using a gradient-descent-based algorithm the weights of the network can be updated in order to find the ground state energy. By the end of the training, the network is expected to converge to a quantum state that is close enough to the actual ground state of the system.

NQSs can be formulated in a symmetry-conserving fashion so that the quantum states generated by the network respect the different symmetries of the problem (20). Moreover, there are indications that NQSs can compress the relevant information of many-body wavefunction which allows us to simulate the behavior of these complex systems (1; 21; 22). The system studied in this project is fermionic, thus, it is anti-symmetric with respect to permutations by nature. Therefore, we must implement the symmetry-conserving property of the NQS to encode these symmetries into the network.

Now we will discuss the NQS proposed by Ref. (7), where they introduced an anti-symmetric NQS as an ansatz for the wavefunction. In the following section, we will explain the architecture and structure of the network.

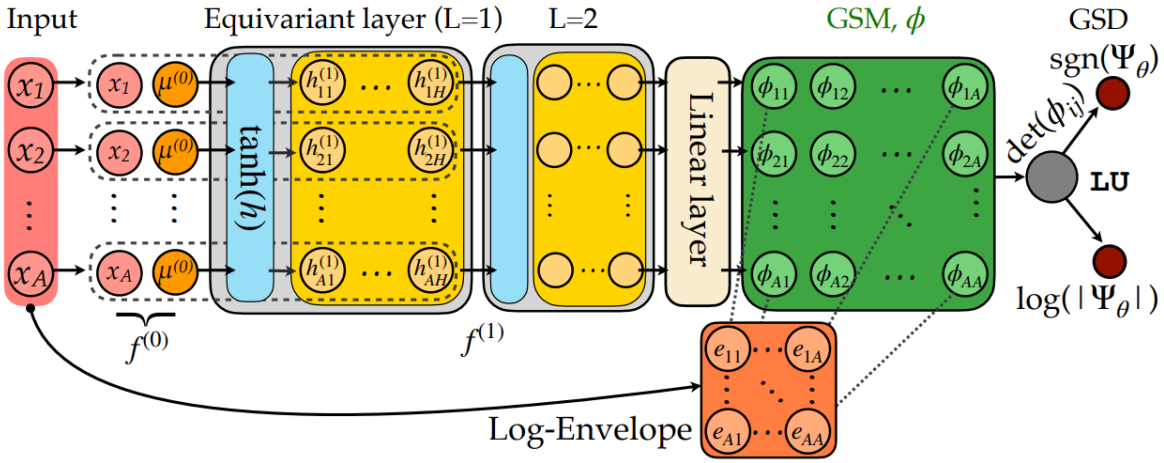


Figure 1: The NQS ansatz of this work for $L = 2$ equivariant layers of H hidden nodes. The input to the network is the set $\{x_i, i = 1, ..., A\}$ of particle positions. These are processed by 2 equivariant layers (light gray areas), to ensure that the NQS maintains equivariance throughout its forward pass. The grey dashed lines denote all nodes involved in the linear equivariant mapping, which is subsequently shared over all input nodes and passed through a hyperbolic tangent function (blue area). The final embeddings are projected via a linear layer into an $A \times A$ matrix (green area), which is element-wise multiplied by the log-envelope layer (orange area) before taking a determinant. In the final step, an LU decomposition is used to find the sign and logarithm of the absolute value of the many-body wavefunction, Ψ. This figure is taken from Ref. (7).

2.3.1 Ansatz Architecture

The design of the ansatz used in this project is inspired by FermiNet (4). FermiNet was able to preserve the anti-symmetry of the network by adding layers of generalized Slater matrices (GSMs). The network consists of A input nodes, each containing the position of one of the A particles in the system. The architecture of the NQS ansatz consists of four main elements: the equivariant layers, the generalized Slater matrices (GSMs), log-envelope functions, and a summed signed-log determinant function. This NQS structure follows the one proposed by ref. (7) and it can be seen in Fig. 1. Then the output nodes of the network give the logarithm of the wavefunction of the system alongside the sign of the wavefunction.

The equivariant layers are used to enforce the permutation equivariance across the network, which helps ensure the anti-symmetry is respected in the network. This means that any permutation done on the input variables is reflected by a permutation on the output of these layers. The input data is pre-processed by adding a permutation-invariant feature, which in this case is the mean many-body position, $\mu^{(0)} = \frac{1}{A}\sum_{i=1}^{A} x_i$.

The first equivariant layer (left gray box in Fig. 1) takes in an input feature $f^{(0)} \in \mathbb{R}^{A\times 2}$ which is a tensor of shape $[A, 2]$ where the first dimension stores the position of the particle i and the second stores the mean position $\mu^{(0)}$,

$$f_{ij}^{(0)} = \begin{cases} x_i & j = 1 \\ \mu^{(0)} & j = 2 \end{cases}. \tag{4}$$

The output of this layer is an H-dimensional representation of the positions $h^{(1)} \in \mathbb{R}^{A\times H}$ (yellow boxes in Fig. 1), where H is the number of hidden neurons (i.e. number of neurons in the hidden layers). This layer consists of a linear transformation using the weights and

biases, followed by a non-linear activation function. Each row $h_i^{(1)} \in \mathbb{R}^{1\times H}$ in the output of this layer is defined as

$$h_i^{(1)} = \tanh\Big(f_i^{(0)} W^{(1)} + b^{(1)}\Big), \tag{5}$$

where i represents the row index (i.e. the particle index). Moreover, $W^{(1)} \in \mathbb{R}^{2\times H}$ and $b^{(1)} \in \mathbb{R}^{1\times H}$ are the weights and biases of this layer, respectively. Note that the shapes of both the weights and the biases in this layer are independent of the number of particles A. This means that as the number of particles changes, the shape of the weights and biases does not change. Nevertheless, as A becomes of the same order as the number of hidden neurons H then we may need to increase H to allow the NQS to accurately represent the wavefunction. This is because the model becomes more complex and requires more variational parameters to maintain its accuracy.

The second equivariant layer of the model takes two inputs: the output of the first equivariant layer, alongside its column-wise averages. The column-wise averages of the output of the first equivariant layer $\mu^{(1)}$ have the shape $\mu^{(1)} \in \mathbb{R}^{H}$. Then the input of the second equivariant layer $f^{(1)}$ is defined as

$$f_{ij}^{(1)} = \begin{cases} h_{ij}^{(1)} & j \leq H \\ \mu^{(1)} & j > H \end{cases}, \tag{6}$$

where the shape is $f^{(1)} \in \mathbb{R}^{A\times 2H}$. Similarly to the first equivariant layer, this layer's weights $W^{(2)} \in \mathbb{R}^{2H\times H}$ and biases $b^{(2)} \in \mathbb{R}^{1\times H}$ are independent of the number of particles A. The output of this layer $h^{(2)} \in \mathbb{R}^{A\times H}$ is composed of A rows, where each row $h_i^{(2)} \in \mathbb{R}^{1\times H}$ has the form:

$$h_i^{(2)} = \tanh\Big(f_i^{(1)} W^{(2)} + b^{(2)}\Big) + h_i^{(1)}. \tag{7}$$

Following this second equivariant layer is a linear layer (beige area in Fig. 1), which takes as input the $A \times H$ matrix generated by the equivariant layers. This layer has shared weights $W^{(M)} \in \mathbb{R}^{H\times A}$ and biases $b^{(M)} \in \mathbb{R}^{1\times A}$ for all the rows of the output matrix $M \in \mathbb{R}^{A\times A}$. Each row M_i can be expressed as:

$$M_i = h_i^{(2)} W^{(M)} + b^{(M)}. \tag{8}$$

Moreover, to impose the asymptotic boundary conditions of the wavefunction decaying at infinity, an envelope function $e \in \mathbb{R}^{A\times A}$ is used (red box in Fig. 1). For best numerical stability this function is implemented in the log domain and has the form:

$$\ln(e_{ij}) = -(x_i W_j^{(e)})^2, \tag{9}$$

where i represents the index for the particle, j is the index of the natural orbital corresponding to the single-particle state, and $W^{(e)} \in \mathbb{R}^{A}$ are the weights associated with the log-envelope of each orbital. The element-wise product between the envelope and M results in a GSM, $\phi \in \mathbb{R}^{A\times A}$ (green box in Fig. 1). Each element of this matrix is given by,

$$\phi_{ij} = M_{ij} * e_{ij}. \tag{10}$$

The final step is to take the determinant of the GSM to obtain an anti-symmetric wavefunction. This determinant is usually known as the generalized Slater determinant (GSD). Note that the GSD is computed within the log domain for numerical stability.

In this project, we use two equivariant layers and we set the number of hidden neurons to $H = 64$. The total number of variational parameters n for the network with two equivariant layers is given by

$$n = 4H + 2A + 2H^2 + H \times A. \tag{11}$$

Therefore, for 4 particles, the total number of parameters required to train the network is $n = 8712$ parameters. If H is larger than A, then the biggest bottleneck are the equivariant layers which have a quadratic dependence on H and a total of $4H + 2H^2$ parameters. For instance, in the case of 4 particles, these equivariant layers account for 8448 parameters out of the 8712. Additionally, if we increase the number of particles to 6 the total number of parameters will become 8844 which is only 132 parameters more than the 4 particles case.

In terms of the memory size of the network, this value depends quadratically not only on H but also on A. This quadratic A dependence comes from the GSM and log-envelope layers, while the quadratic H dependence comes from the equivariant layers.

2.3.2 Metropolis-Hastings Samplings

To calculate the expectation value of the energy, we use a Monte-Carlo estimation of the integral in Eq. (3). For this estimation, we need to sample the position values of the particles according to the squared amplitude of the parameterized wavefunction $|\Psi|^2$, represented by the network, which can be done using a sampling technique.

We use the Metropolis-Hastings (MH) sampling (23) to obtain samples for the positions of the particles $\mathbf{x} = (x_1, x_2, ..., x_A)$ according to the distribution of the wavefunction. The MH sampling is a Markov chain Monte Carlo (MCMC) method that generates a sequence of samples based on a proposal distribution and an acceptance criterion (24).

MH is a specific type of MCMC algorithm that uses a proposal distribution P to generate a candidate sample $\mathbf{x}^{(t)}$ at iteration t. The candidate sample is then accepted or rejected based on an acceptance criterion. The acceptance criterion involves computing the ratio of the target distribution G at the current and candidate samples, $r = \frac{G(\mathbf{x}^{(t+1)'})}{G(\mathbf{x}^{(t)})}$, where $\mathbf{x}^{(t+1)'}$ denotes the candidate sample for the following iteration. If $r \geq 1$, the candidate sample is always accepted. If $r < 1$, the candidate sample $\mathbf{x}$ is accepted with a probability equal to the ratio r.

In this project, we start with multiple walkers randomly distributed across the space. Then, each walker evolves according to the MH algorithm for a specified amount of steps (or sweeps). Finally, the samples are taken to be the final value of each walker, and the rest of the steps taken by the walkers are discarded.

This process allows us to sample a set of points from the NQS wavefunction, which we can then use to compute the energy. For instance, suppose that a set of configurations $\mathbf{x}^{(i)}$ where $i = 1, ..., N_{samples}$ has been sampled from the modulus squared of the variational state $|\Psi_\theta(\mathbf{x})|^2$. Then, the expectation value of the energy, E_θ, can be calculated using the parameterized wavefunction by,

$$E_\theta = \frac{\int d\mathbf{x} \Psi_\theta^\dagger(\mathbf{x}) \hat{H} \Psi_\theta(\mathbf{x})}{\int d\mathbf{x} |\Psi_\theta(\mathbf{x})|^2} \approx \mathbb{E}_{\mathbf{x} \sim |\Psi_\theta(\mathbf{x})|^2}[e_{L,\theta}(\mathbf{x}^{(i)})], \tag{12}$$

where $\mathbb{E}_{\mathbf{x}^{(i)} \sim |\Psi_\theta(\mathbf{x})|^2}$ is the statistical expectation over the $N_{samples}$ samples, Ψ_θ is the parameterized NQS and θ represents the network parameters (10). Moreover, $\hat{H}$ is the Hamiltonian of the system, and the local energy is expressed as $e_{L,\theta}(\mathbf{x}^{(i)}) = \frac{\langle \mathbf{x}^{(i)} | \hat{H} | \Psi_\theta \rangle}{\langle \mathbf{x}^{(i)} | \Psi_\theta \rangle}$. Within statistical uncertainties, the variational principle ensures that E_θ is larger than the ground state energy, E_{gs}.

Moreover, the variance in the energy calculation can be estimated by,

$$\sigma^2_{\Psi_\theta(\mathbf{x})} = \frac{\int d\mathbf{x} \Psi_\theta^\dagger(\mathbf{x}) (\hat{H} - E_\theta)^2 \Psi_\theta(\mathbf{x})}{\int d\mathbf{x} |\Psi_\theta(\mathbf{x})|^2} \approx \mathbb{E}_{\mathbf{x} \sim |\Psi_\theta(\mathbf{x})|^2}[e_{L,\theta}(\mathbf{x}) - E_\theta]^2. \tag{13}$$

The optimization of the network is done by changing the variational parameters θ to minimize the energy E_θ, so that it is as close as possible to the ground state energy E_{gs}. In this project, we use 4096 independent walkers to perform the MH sampling and 10 steps per walker, unless otherwise stated.

2.3.3 Loss Function and Training

In this section, we describe the training process of the network. The critical element in this process is the loss, as it is what the network uses to optimize the variational parameters. The local energy is given by,

$$e_{L,\theta}(\mathbf{x}) = -\frac{1}{2}\sum_{i=1}^{A}\left[\frac{\partial^2 \ln|\Psi_\theta(\mathbf{x})|}{\partial x_i^2}\bigg|_{\mathbf{x}} + \left(\frac{\partial \ln|\Psi_\theta(\mathbf{x})|}{\partial x_i}\right)^2\bigg|_{\mathbf{x}}\right] + \sum_{i=1}^{A}\frac{x_i^2}{2} + \frac{V_0}{\sqrt{2\pi}\sigma_0}\sum_{i<j} e^{-\frac{(x_i-x_j)^2}{2\sigma_0^2}}, \tag{14}$$

where the kinetic term was computed in the log domain for numerical precision. The loss that we implement is expressed by the following equation,

$$\mathcal{L}(\theta) = \frac{1}{\perp\left[\sum_i^{N_{samples}} R_\theta(\mathbf{x}^{(i)})\right]} \sum_i^{N_{samples}} \perp\left[R_\theta(\mathbf{x}^{(i)})(e_{L,\theta}(\mathbf{x}^{(i)}) - E_\theta)\right] \ln|\Psi_\theta(\mathbf{x}^{(i)})| \tag{15}$$

where E_θ is given by Eq. (12), and $R_\theta(\mathbf{x}^{(i)}) = 1$ if the MH sampling happens at every training step. In case the samples are reused for some epochs before updating, then R_θ will change to account for the energy correction due to reusing the samples. This idea will be further discussed in section 3.3 and there a more general and complete expression for $R_\theta(x)$ will be provided.

Moreover, $\perp$ () is the detach function (25). This function is the identity function except that its derivative equals 0, i.e. $\perp(x) = x$ but $\frac{\partial \perp(x)}{\partial x} = 0$. This restricts the gradient of the loss to only flow back through $\ln(\Psi_\theta)$, which is important for the automatic differentiation step so the gradient of the network's weights can be properly calculated as $\partial_\theta E_\theta = \partial_\theta \mathcal{L}(\theta)$. This ensures that the loss function can be computed avoiding higher-order derivatives.

The optimizer chosen for this project is Adam (26) with a learning rate of 1×10^{-4} (for more details the reader is asked to refer to appendix B Sec. B.7). Moreover, the network hyper-parameters are fixed. As mentioned before, the number of equivariant layers was set to $L = 2$, and the number of hidden nodes was set to $H = 64$. These values were selected as they show optimal results according to Ref. (27). Moreover, the networks trained in this project were allowed to run for a maximum of 10^5 epochs. This cut-off was defined to set an upper limit to the training time.

3 Implementation

This section will explain the different machine learning techniques used in this project and provide a summary of the improvements each provided to the network. The first technique we will cover in this section is Early Stopping (Sec. 3.1), which provides a method to detect when the network has converged, thus, being able to stop the training process. Then, we will go over Transfer Learning (Sec. 3.2) and the different ways that we were able to exploit this technique to speed up the learning of the network. Finally, we discuss the ways that are found to speed up the Metropolis-Hasting process (Sec. 3.3) which we identified to be the slowest part of the training.

Note that all of the plots shown in this and the following sections are using networks trained for $A = 4$ particles, $V_0 = 10$, and $\sigma_0 = 0.5$. We chose this interaction strength because it is a non-trivial example in the repulsive regime for which we had the data for the exact diagonalization of the Hamiltonian so we could check the performance of the model.

3.1 Early Stopping

The idea of Early Stopping is used to stop the training of the network when a specific criterion is satisfied (8). This technique is a very valuable regularization method that is used to prevent the network from overfitting. Early Stopping also helps improve the generalizability of the neural network (28).

The most important aspect of Early Stopping is defining the criterion at which the learning should be stopped. In the case of this project, the criterion had to include different aspects of the energy evolution to ensure that the mean value was not going to change throughout the remaining epochs. To inspect the convergence of the model three main measures were set in place. These measures are evaluated on a periodic basis using a pre-defined window of values taken from the last epochs of training.

For each window i of 1000 epochs we first check whether the slope of the energy evolution S_i in this window is small enough, which indicates a plateau in the network's training. This slope is calculated by fitting a line to the energy values within the window i. Additionally, the slope in the previous window S_{i-1} needs to be small also, and their difference needs to be within a specific threshold. If these conditions are satisfied, then we proceed to check the average value of the energies within the current window $\mathbf{E}_i$. This value needs to be close enough to the average of the energies in the previous window $\mathbf{E}_{i-1}$. Furthermore, the average of the variance V_i of each energy point in the current window and the standard deviation of the distribution of energy points within the window D_i needs to be reasonably small. Once all of these conditions are satisfied four consecutive times then we can confidently say that the network has converged. We allow two errors to occur between these 4 consecutive successes to permit for a more flexible convergence.

To quantify how small threshold for S_i, E_i, V_i, and D_i should be we ran different tests and selected the ones that provided reasonable convergence within the given problem. The values of these thresholds are selected depending on the desired accuracy in the energy calculation. This accuracy can be evaluated based on the uncertainty in the energy calculation using the Monte Carlo integral. For 4 particles, we use a threshold of 3×10^{-7} to check that the slopes S_i at the window i are flat, and to compare how different the slopes are from each other at two consecutive windows we enforce that $|S_i - S_{i-1}| < 3 \times 10^{-6}$. For the energy difference $|\mathbf{E}_i - \mathbf{E}_{i-1}|$ we use a threshold of 10^{-3}. Similarly, we chose a threshold of 3×10^{-3} for both the average energies variance V_i and the standard deviation of the energy points in the window D_i.

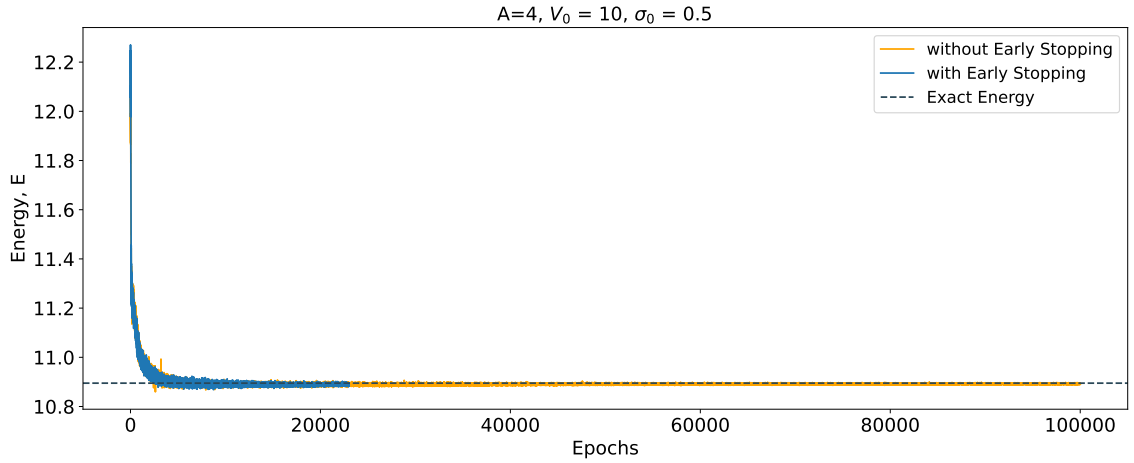


Figure 2: The energy as a function of the training epochs. The blue solid line has Early Stopping active while the orange line does not. The black dashed line represents the exact value of the ground state energy computed using the exact diagonalization of the Hamiltonian from Ref. (12).

For more details refer to Algorithm 1 in Appendix A which describes how the procedure of Early Stopping is carried out.

Fig. 2 shows a comparison between the training process of two networks. One of the networks is trained with Early Stopping active (blue solid line), while the other is trained without Early Stopping (orange solid line). The results show that the training with Early Stopping stops when the network reached a plateau. This process only takes 23022 epochs, while when Early Stopping is not active the network will train until the maximum epoch we set which is 10^5. In the epochs between the epochs when the network with Early Stopping converged and the final epoch, the energy changes are minimal. For instance, the energy returned by the network with the Early Stopping is $E = 10.89 \pm 2 \times 10^{-2}$ while the network that trained for longer returned a final energy value of $E = 10.892 \pm 5 \times 10^{-3}$. The error in the energy calculation is computed using Eq. (13). It is important to keep in mind that the exact value of the energy computed with exact diagonalization of the Hamiltonian is $E = 10.8916$ which means that both networks returned valid energy values within their respective error margins. The precision desired to be obtained from the network can be factored into the Early Stopping thresholds, but there is a trade-off between how restrictive the thresholds are and the time it takes for the Early Stopping to trigger.

3.2 Transfer Learning

Transfer Learning is a research problem in the machine learning field that concerns applying the knowledge gained in training one task to another related task (9). For example, knowledge gained while learning to recognize cars could be applied when trying to recognize trucks. What makes this methodology very useful is that generally only a few samples from the new task are needed to re-purpose a model. Therefore, when there is a lack of data points, Transfer Learning can exploit the information learned from one task to improve the generalization in another (29).

Transfer Learning works by transferring the weights of a network after being trained with task A to another network used to learn task B. This means that instead of starting the learning process from scratch, the network can start with patterns learned from solving a related task.

In this project, we have used Transfer Learning in two different ways. The first is by

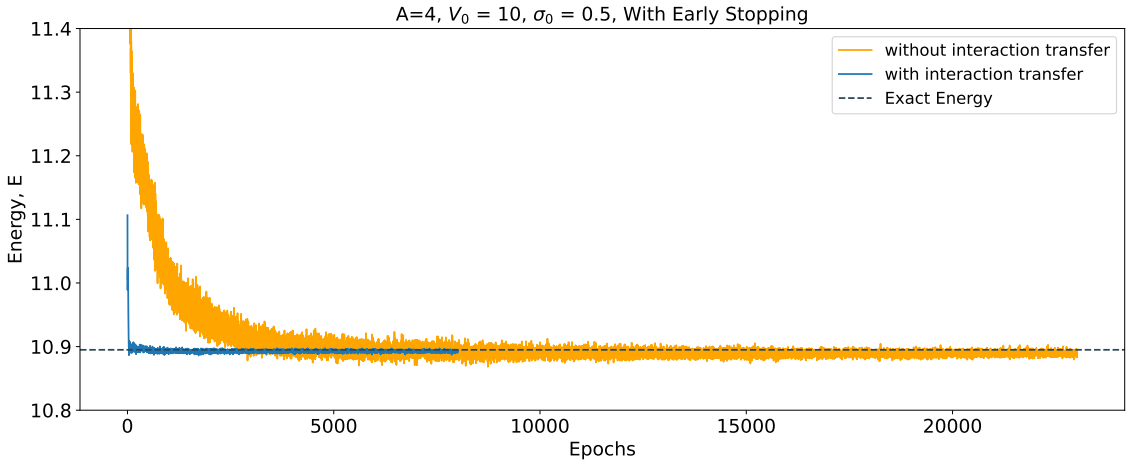


Figure 3: The energy as a function of the training epochs. The blue solid line represents a minimization with Interaction Transfer, while the orange solid line represents one without Interaction Transfer (same as the blue line in Fig. 2). In both cases, Early Stopping is active. The dark dashed line represents the exact value of the ground state energy computed using the exact diagonalization of the Hamiltonian.

moving the weights of a network trained with one interaction to another network that will be trained with a different interaction. The second is by moving the weights of a network trained with fewer particles, to another with a larger number of particles. In the second case, both networks are trained with the same interaction and the only change is in the number of particles.

3.2.1 Interaction Transfer

In the interaction transfer case, the network is first trained for initial values of V_0 and σ_0. Then, the trained parameters of this network are used as the initial guess when training the network with target V_0' and σ_0' values. The benefits of Interaction Transfer can be seen clearly in Fig. 3. In this figure, we compare the energy minimization as a function of the training epochs for two networks. These two networks are trained for $A = 4$ particles, at the target interaction values $V_0 = 10$, and $\sigma_0 = 0.5$. Moreover, both networks are trained with Early Stopping active to showcase the advantages that Interaction Transfer provides in terms of convergence. When Interaction Transfer is active we first pre-train the network for the interaction $V_0 = 5$ and $\sigma_0 = 0.5$, then the weights produced by this pre-trained network are used as the initial weights for the final network. The blue line represents the network with Interaction Transfer active, while the orange line represents the network without Interaction Transfer. The network with Interaction Transfer converges within 8007 epochs, with an energy of $E = 10.892 \pm 4 \times 10^{-3}$.

As shown in the previous section, the network without Interaction Transfer needs about 23022 epochs to converge and has an energy of $E = 10.89 \pm 2 \times 10^{-2}$. This means that not only does the network with Interaction Transfer converge more quickly, but it also estimates the energy more accurately.

3.2.2 Particle Transfer

On the other hand, in the number of particles Transfer Learning, the network is trained initially with a number of particles lower than the target number of particles. However, it is important to note that as the number of particles A grows, the size of the network grows

as well, which means that not all of the network parameters can be transferred when we move to a higher number of particles A'. Nevertheless, the equivariant layers' size does not depend on the number of particles (which are the light gray areas in Fig. 1). This implies that as the number of particles A increases (up to a certain value) the size of the equivariant layers does not change (In the case of $A = 4$ particles this accounts for 8448 parameters out of the total 8712 parameters), which allows it to be completely transferred to the network solving the interaction with the target number of particles A'.

Moreover, as this part of the network does not change in size with the change in the number of particles, we can demonstrate that even if we first train this network for fewer particles, then transfer the weights of these layers to a network with a higher number of particles. Even if we freeze the training of the parameters in this part of the network after the transfer of the parameters, the network can still converge to the minimum value. In this context, freezing a part of the network means fixing the values of the parameters corresponding to that part and preventing them from being updated during training (30), which can reduce the memory and computational cost of the training process.

In Fig. 4, we compare the training of a network with Particle Transfer (blue line) and another without Particle Transfer (orange line). In this case, Particle Transfer was performed by first training a network with $A = 2$ particles, then moving the compatible weights to the network with $A = 4$ particles. Moreover, we froze the training on the layers where the weights were transferred. Both networks have Early Stopping active to compare convergence. However, as seen in the figure, the network with the Particle Transfer does not trigger the Early Stopping because the energy minimization remains noisy throughout the training.

We spectate that it is still important to pay attention to the change in the number of trainable parameters. For instance, without freezing the equivariant layers, the network has 8712 trainable parameters for the $A = 4$ particles case. After performing the freezing, the number of trainable parameters drops to 264 parameters. Moreover, as the number of trainable parameters is reduced, the time it takes to perform the backward propagation and the optimization step should also be reduced. This means that this approach is supposed to not only provide memory improvements (reduces the weights needed for training by 98%) but also temporal improvements as well.

Fig. 5 provides a full time-profiling of the training. In this case, we deactivated Early Stopping as we wanted to compare both networks until the maximum epoch. The blue and orange points correspond to networks trained with or without Particle Transfer respectively. The temporal advantage that the freezing provides can be seen in the optimization (Fig. 5.b) and backward pass times (Fig. 5.c). This is because freezing the equivariant layers reduces the number of parameters drastically, which in turn decreases the time it takes to calculate the gradients and update the weights. Nevertheless, the temporal improvement of this approach is almost negligible. This is because of the time it takes to perform the MH sampling (Fig. 5.e), which is an order of magnitude larger than the time it takes to perform the optimization and the backward pass combined. This limitation prevents the freezing of the equivariant layers from providing any tangible benefits.

3.3 Scheduled Re-weighted Metropolis-Hastings

As noted before, the MH sampling takes an order of magnitude longer to execute at each training step. However, this process does not need to be executed at every time step, as the probability distribution is expected to change slowly across training. Each sampling process can be reused for a few training steps before needing to update it again to match the current wavefunction.

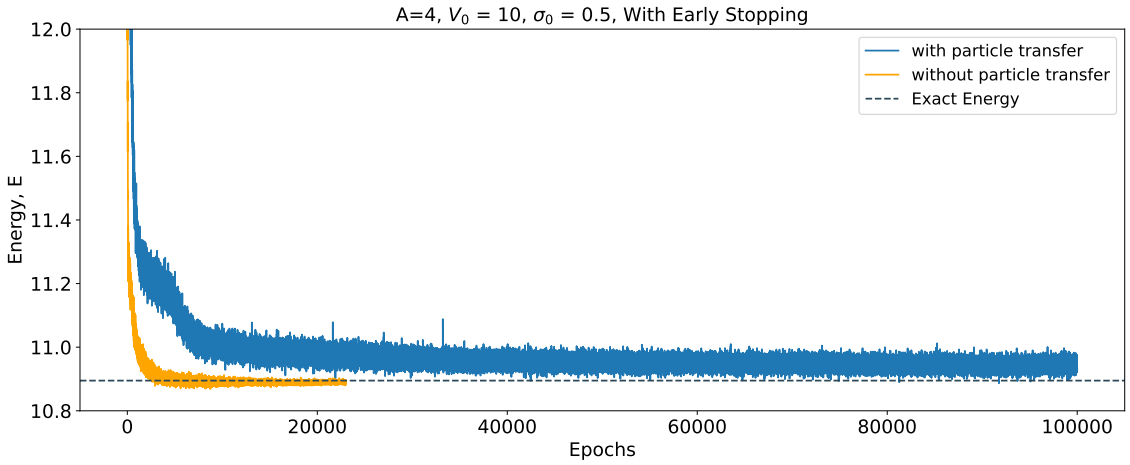


Figure 4: This figure is the same as Fig. 3, but the blue line uses Particle Transfer instead of Interaction Transfer.

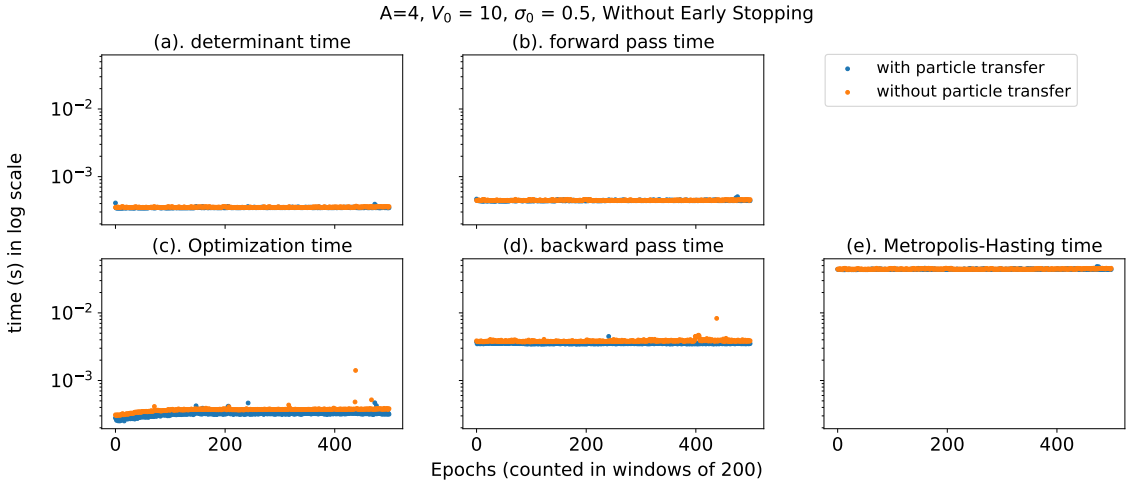


Figure 5: Time profiling of the different sections of the training. This figure compares the time when the network is trained normally (blue), and when it is trained after applying the number of particle Transfer Learning and freezing the equivariant layers (orange). In both cases, Early Stopping is deactivated to compare the speed of execution across the full training. (a) shows the time it takes the network to compute the determinant after the GSM and log-envelope layers are applied. (b) shows the time it takes to perform a forward pass on the network. (c) shows the time it takes the optimizer to compute gradients that will be used to update the weights. (d) shows the time it takes to perform the backward pass and update the network weights. (e) shows the time it takes the MH sampler to compute the samples according to the wavefunction.

To do this as accurately as possible, we use a re-weighting method to measure the similarity between the original values of the sampled points and the new values according to the new wavefunction. This ratio is given by

$$R_{\theta+\delta\theta}(\mathbf{x}) = \frac{|\Psi_{\theta+\delta\theta}(\mathbf{x})|^2}{|\Psi_{\theta}(\mathbf{x})|^2}, \tag{16}$$

where Ψ_θ is the wavefunction at the epoch when the values of $\mathbf{x}$ were sampled, and $\Psi_{\theta+\delta\theta}$ is the wavefunction at the current epoch, with slightly different variational parameters $\theta + \delta\theta$.

As the samples used to calculate the expectation value of the energy are distributed according to an older wavefunction, some energy correction terms need to be added to the expected value calculations. This technique is known as re-weighting or importance sampling the MCMC literature (10). For instance, the new expected energy in Eq. (12) can be computed using,

$$E_{\theta+\delta\theta} = \frac{\mathbb{E}_{\mathbf{x}\sim|\Psi_\theta(\mathbf{x})|^2}(e_{L,\theta+\delta\theta}(\mathbf{x})R_{\theta+\delta\theta}(\mathbf{x}))}{\mathbb{E}_{\mathbf{x}\sim|\Psi_\theta(\mathbf{x})|^2}(R_{\theta+\delta\theta}(\mathbf{x}))}. \tag{17}$$

Moreover, the variance in the energy specified in equation 13 now becomes,

$$\sigma^2_{\Psi_{\theta+\delta\theta}} = \frac{\mathbb{E}_{\mathbf{x}\sim|\Psi_\theta(\mathbf{x})|^2}([e_{L,\theta+\delta\theta}(\mathbf{x}) - E_{\theta+\delta\theta}]^2 R_{\theta+\delta\theta}(\mathbf{x}))}{\mathbb{E}_{\mathbf{x}\sim|\Psi_\theta(\mathbf{x})|^2}(R_{\theta+\delta\theta}(\mathbf{x}))}. \tag{18}$$

Ideally, the mean of the ratio $R_{\theta+\delta\theta}$, which is defined as

$$\overline{R_{\theta+\delta\theta}} = \mathbb{E}_{\mathbf{x}\sim|\Psi_\theta(\mathbf{x})|^2}(R_{\theta+\delta\theta}(\mathbf{x})), \tag{19}$$

should be 1. But as the training progress evolves, this effective sample size starts to decay from 1 and the samples obtained using MH become less and less representative of the wavefunction at the current epoch.

To account for this decay in $\overline{R_{\theta+\delta\theta}}$, only a limited number of epochs between 1 and 100 are allowed to run before a new MH calculation of the samples is performed. The number of epochs before the next MH calculation is determined by the following schedule,

$$num\_waited\_epochs = upper\_limit - (upper\_limit - lower\_limit) * \Delta, \tag{20}$$

where $\Delta = |1 - \frac{1}{N_{samples}} \sum_{i=1}^{N_{samples}} e^{-\frac{(R_{\theta+\delta\theta}(x_i)-1)^2}{s}}|$ and $s = 10^{-2}$. For this project, we set the *upper_limit* to 100 and the *lower_limit* to 1.

Fig. 6 shows the energy minimization of a network with Scheduled Re-weighing MH (blue line), while the other network (orange line) was trained sampling at each epoch. Both networks in this case have Early Stopping active to compare convergence. We can see in the plot that the network with re-weighting learns faster and manages to converge faster. However, this is not the case always as we will see in Sec. 4. The blue network takes 15014 epochs to coverage and has an energy of $E = 10.89 \pm 1 \times 10^{-2}$. Whereas the orange network is the same network as the blue network seen in Fig. 2.

Fig. 7 it shows a full time-profiling of the energy minimization of both blue and orange networks from Fig. 6. In this case, we turned off Early Stopping on both networks to be able to conduct a fair comparison. The MH sampling time is reduced by about a factor of 10. The rest of the network time profilings had minimal to no change when Scheduled Re-Weights MH is applied as expected.

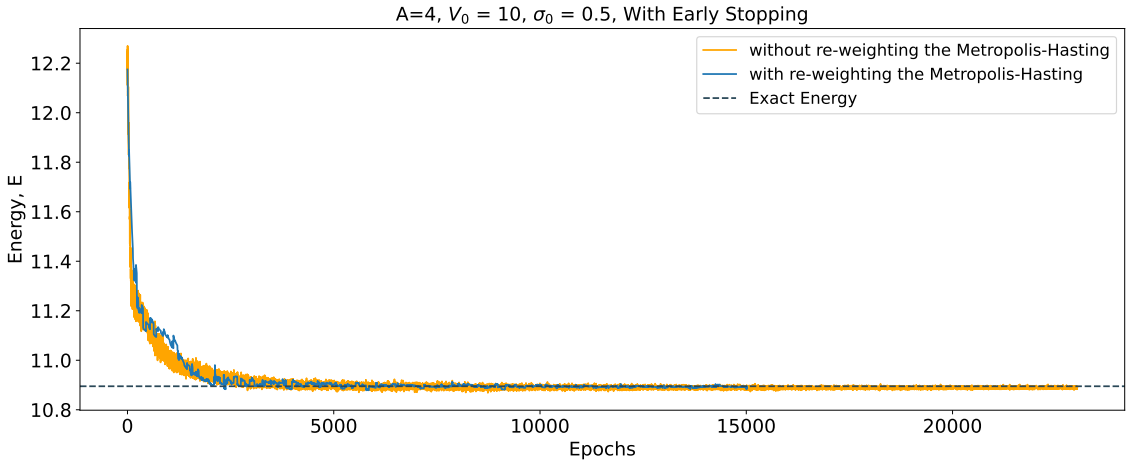


Figure 6: This figure is the same as Fig. 3, but the blue line uses Scheduled Re-Weighted MH and does not use any type of Transfer Learning.

The number of epochs waited by the network before re-sampling is shown in Fig. 8. In this figure, the orange line represents the evolution of the value of the schedule calculated using Eq. 20. The blue line shows the number of epochs the network has waited since the last re-sampling. When the orange line and the blue line cross is when the re-sampling happens. Fig. 9 shows a zoomed-in version of Fig. 8 that allows us to clearly see how the schedule evolves at each epoch. This indicates that the scheduler advises for a re-sampling as the difference between the wavefunction at which the samples were computed and the current wavefunction is growing. The black line shows the average of the waited epochs in windows of 5 epochs. This shows how the scheduling adapts to the training progress. For example, we can observe that the schedule increased the network re-sampling rate in the early stages of the training. This makes sense because the network is far from the ground state wavefunction at the start and needs more frequent updates. But as the training goes on, the network gets closer to the ground state wavefunction and does not require as much re-sampling as before. For instance, toward the beginning of the training the network re-sampled on average every 20 epochs. Toward the end of the training the network samples on average every 80 epochs.

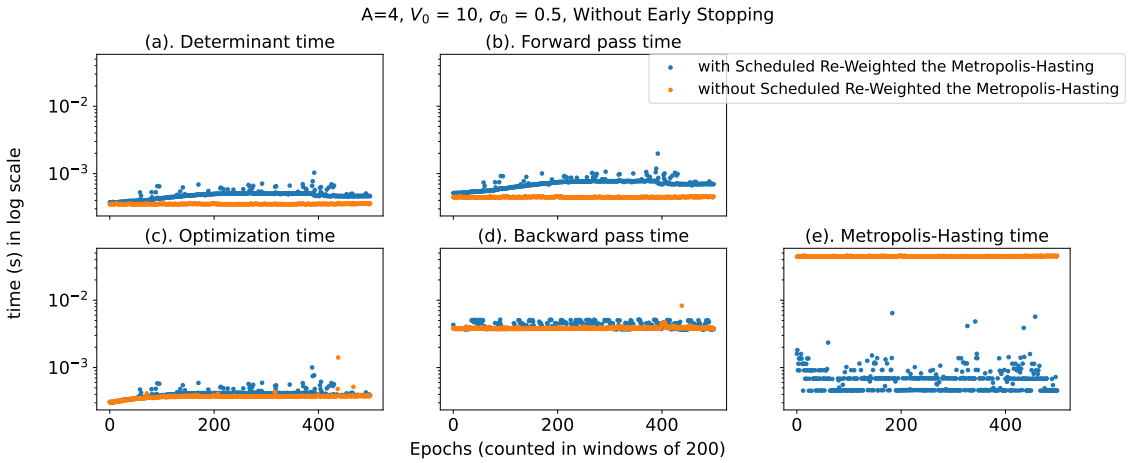


Figure 7: This figure is the same as Fig. 5 but the blue scatter represents the network with Scheduled Re-Weighted MH without any Transfer Learning active on it. While the orange scatter is the same, and it doesn't have any of the techniques discussed active during the training.

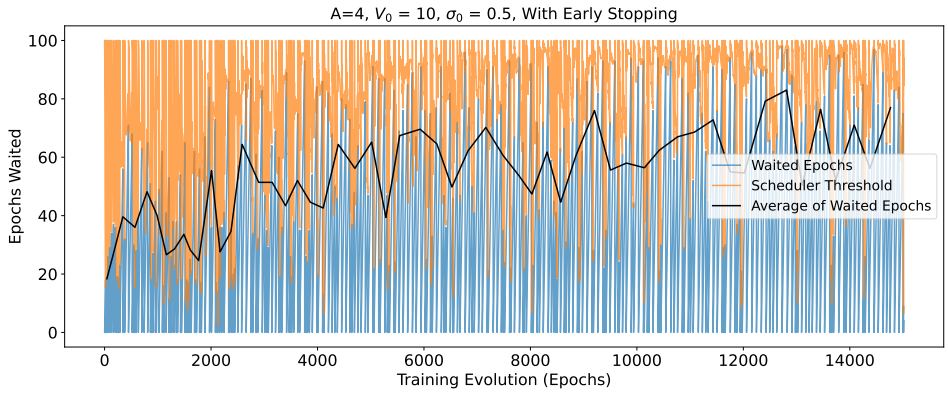


Figure 8: The number of epochs waited by the network before updating the MH samples (blue line) in contrast to the value of the threshold calculated using equation 20 (orange line). Moreover, the black line shows the average number of waited epochs.

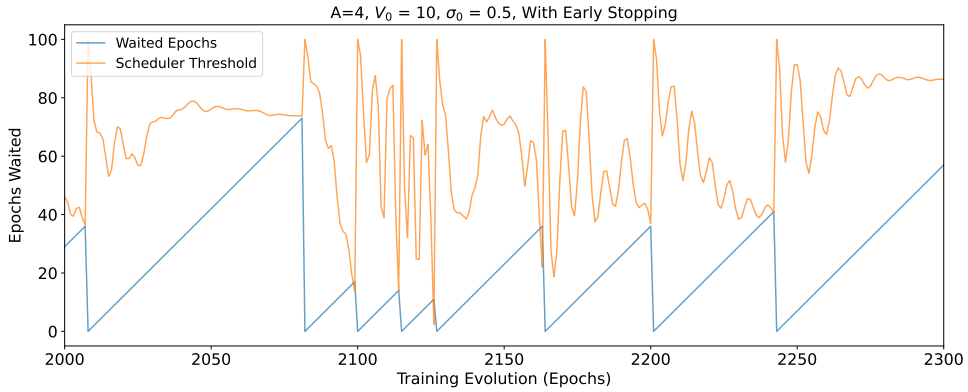


Figure 9: A zoom in between epochs 2000 and 2300 from Fig. 8.

4 Results

This section will concern the overall outcomes of the research and will try to combine the different approaches explained in previous sections to achieve optimal improvement in terms of convergence and time.

4.1 Particle and Interaction Transfer

As mentioned earlier, Particle Transfer tries to transfer parts of a network prepared with the target interaction but for fewer particles, to a network with more particles. Interaction Transfer does not change the number of particles but moves the information of a network trained for an initial interaction to another network that is being trained with the target interaction.

Building a test that encompasses both of them is not trivial, as they both transfer the network's information in a different manner. This is why this test had multiple steps. The target network is a network with $A = 4$ particles, $V_0 = 10$ and $\sigma_0 = 0.5$. The test first starts by training a network with $A' = 2$ particles and $V_0' = 5$ and $\sigma_0' = 0.5$. Then, this network is transferred using Particle Transfer into a network with $A = 4$ particles, $V_0' = 5$ and $\sigma_0' = 0.5$. After that, Interaction Transfer is applied to copy the network's weight and use them as a starting point to train with the target interaction $V_0 = 10$ and $\sigma_0 = 0.5$.

One of the advantages of applying Particle Transfer is that it allows the freezing of the equivariant layers of the network. This freezing of the equivariant layers is applied during the Particle Transfer from $A = 2$ to $A = 4$, but it can be disabled in the last step of the training when the final Interaction Transfer is applied. Whether this freezing is activated or deactivated in the final step will have an impact on the network's performance. Fig. 10 shows the effects this freezing has on the network's training. The blue line shows the energy evolution if the freezing is propagated to the final network with $A = 4$ and $V_0 = 10$. The orange line shows the energy training if this freezing is not propagated and the equivariant layers of the final network are allowed to train again.

In this figure, we see that not propagating the freezing does make the training less noisy. However, it is important to note that the network converges in both cases. This means that even though the training can be noisy, it could still be beneficial to propagate this freezing. For instance, when studying a system with a larger number of particles the network size becomes a critical resource to optimize as it can prevent us from being able to execute the network's training.

4.2 Interaction Transfer with Re-Weighted Metropolis-Hastings

It was shown in Sec. 3.2 that Interaction Transfer does provide an improvement in convergence time. But, in Fig. 11 we see that when the scheduled re-weighting of the MH sampling is applied, the calculations overall become less precise. In this figure we have the energy evolution for three networks. All networks were trained with interaction transfer from $V_0' = 5$ to $V_0 = 10$, and they had Early Stopping active during the training. The orange and green lines have Scheduled Re-Weighted MH active while the blue line does not. The green line converged in 19018 epochs and reached an energy of $E = 10.8938 \pm 7 \times 10^{-4}$, while the orange converged in 92091 epochs and reached an energy of $E = 10.892 \pm 3 \times 10^{-3}$. Finally, the blue line converged in 8007 epochs with an energy of $E = 10.892 \pm 4 \times 10^{-3}$. Looking at the plots the spikes in the orange line become immediately noticeable. We have studied the nature of these spikes extensively, and managed to find some ways to mitigate them.

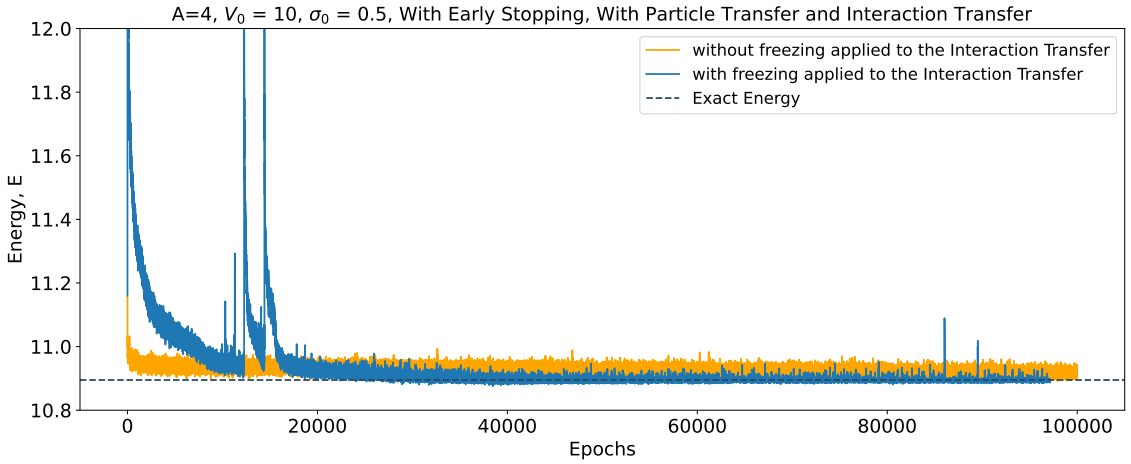


Figure 10: The energy evolution of two different networks with Early Stopping, Particle, and Interaction Transfer active during the training. The network represented by the blue line applies freezing throughout the entire training. The orange line does not apply the freezing when performing the Interaction Transfer. Finally, the dark dashed line represents the exact value of the energy computed by exact diagonalization of the Hamiltonian.

The more concerning spikes are the ones that dip below the ground state energy. This is a consequence of applying the re-weighting technique, as the MC estimation of the integral we do in Eq. (17) is accurate only if the wavefunction $\Psi^{(t)}_{\theta+\delta\theta}$ at epoch t is similar to the original wavefunction $\Psi^{(0)}_{\theta}$ at which the sampling process happened. If the wavefunctions are very different, then the samples don't accurately represent the wavefunction $\Psi^{(t)}_{\theta+\delta\theta}$ at epoch t and that could result in the integral producing wrong results that are bellow the ground state energy. For instance, when the weights are sampled from the original wavefunction $\Psi^{(0)}_{\theta}$, these samples will be reused for up to 100 epochs before sampling again. We have tried to debug the energy evolution to figure out why these spikes are generated if we supposedly re-sample when the networks are relatively different. The hypothesis we have is that the network finds a way to minimize the energy for the given samples, by increasing the value of the wavefunction where the samples are less concentrated and reducing the wavefunction at the other samples. This sudden change in the wavefunction is detected by the schedule a little too late, as the negative spike in energy has already happened. Once this is detected the scheduler forces the network to re-sample. The new samples reveal the real value of the energy, which leads to the positive spike, and then the network corrects its energy minimization over the course of the following epochs.

A way to avoid this sampling issue could be by increasing the number of walkers, and the number of steps each walker takes. Moreover, we can increase the sensitivity of the scheduler in Eq. (20) by changing s in Δ to be 10^{-3} instead of 10^{-2}. As the shape of Δ was chosen to follow that of a Gaussian, reducing s reduces the width of the Gaussian. That allows Δ to drop rapidly to zero when the overlap between the ratio and 1 is lower than 90%. If we want to ensure that the wavefunction never changes a lot we can even provide a hard-coded cutoff for the ratio at which we force the program to re-sample. This will improve the accuracy of the integral estimation and will ensure that the scheduler is reacting to any small changes that happen to the wavefunction $\Psi^{(t)}_{\theta+\delta\theta}$ as soon as they happen. The green line in Fig. 11 shows the improvements in the energy minimization after increasing the number of walkers to 8000 with each walker taking 60 steps. This proves that the problem with the spikes is directly related to the sampling process. For

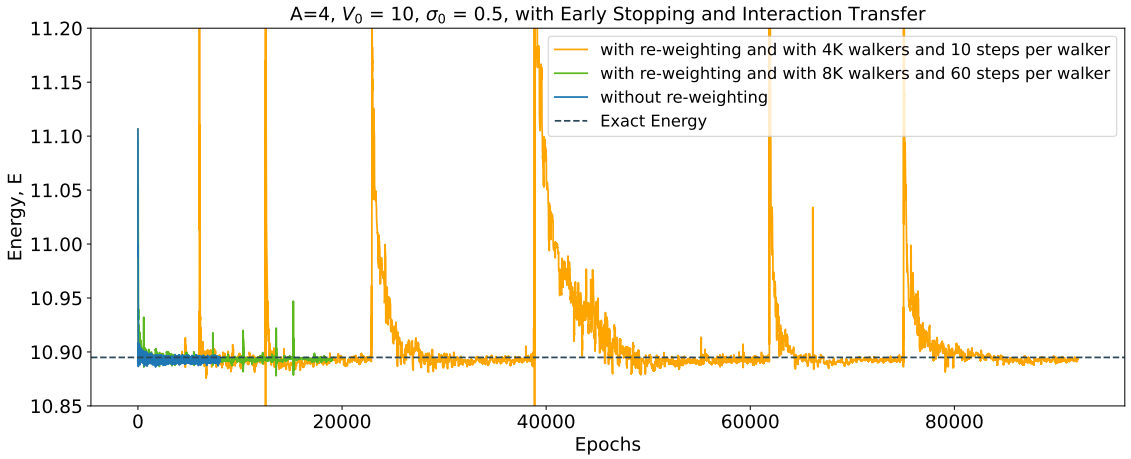


Figure 11: The energy evolution during training for three networks. All the networks are trained for 4 particles, $V_0 = 10$, $\sigma_0 = 0.5$, Early Stopping, and Interaction Transfer. Nevertheless, two of the networks have the Scheduled Re-weighting MH sampling active (orange and green lines), while the other does not. The green network uses 8000 walkers with 60 steps per walker, while the orange uses only 4092 walkers with 10 steps per walker. Finally, the dark dashed line represents the exact value of the energy computed by the exact diagonalization of the Hamiltonian.

instance, when we are increasing the number of walkers they will cover up more of the domain of the wavefunction. This allows us to detect changes in the wavefunctions sooner and more accurately. Moreover, increasing the number of steps per walker reduces the correlation between the samples at this epoch and the samples from the previous epochs. This correlation comes as a result of using the samples from the last epoch as the starting point in the sampling process at the current epoch.

4.3 Particle Transfer with Re-Weighted Metropolis-Hastings

In this section, the Particle Transfer approach is compared with and without the Scheduled Re-Weighted MH sampling. Fig. 12 shows the energy minimization of two networks trained with Particle Transfer, one with Scheduled Re-Weighted MH active (orange line) while the other samples at each epoch (blue line). In this case, we can see that applying the re-weighting technique helped in the energy accuracy. For instance, The network without Re-Weighted MH results in an energy estimation of $E = 10.9 \pm 0.6$, while the network with re-weighting results in an energy value of $E = 10.928 \pm 7 \times 10^{-3}$.

Similar to the case in Sec. 4.2 the energy evolution with Re-Weighted MH does exhibit more spikes which are due to sudden changes in the wavefunction. Moreover, applying the freezing to the network reduces its degrees of freedom, which could make small changes to the variational parameters very prominent in energy.

Nevertheless, the scheduling algorithm seems to correct for it really quickly as it prompts the network to re-sample when the spikes happen and makes sure the energy minimization approaches the ground state energy.

4.4 Particle and Interaction Transfer with Re-Weighted Metropolis-Hastings

As we have seen each of the studied techniques provides an additional advantage to the training. In this section we study the benefits of applying all of these techniques simultaneously.

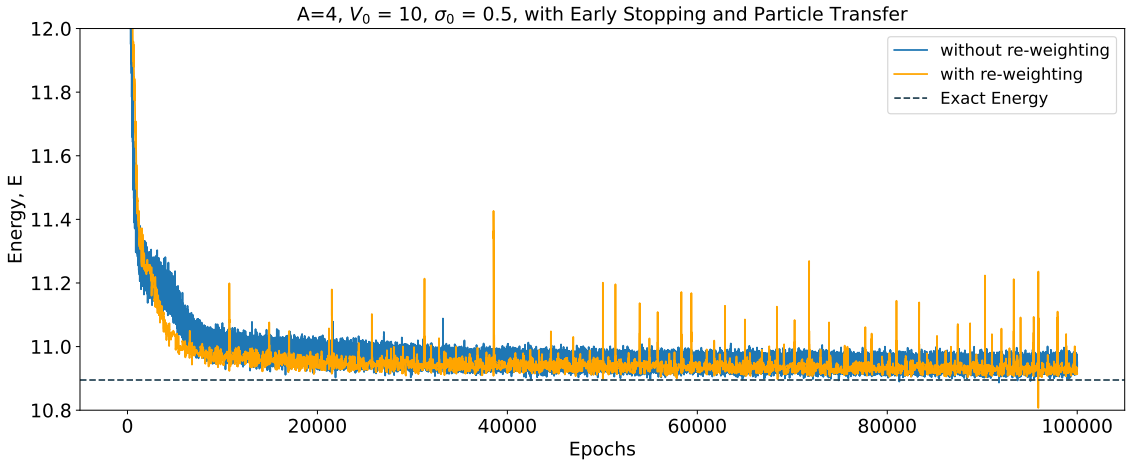


Figure 12: The energy evolution during training for two networks. Both networks are trained with Early Stopping, and Particle Transfer active. Nevertheless, one of the networks has the Scheduled Re-weighting MH sampling active (orange line), while the other doesn't. For the orange line, we used 8000 walkers with 60 steps per walker. Finally, the dark dashed line represents the exact value of the energy computed by the exact diagonalization of the Hamiltonian.

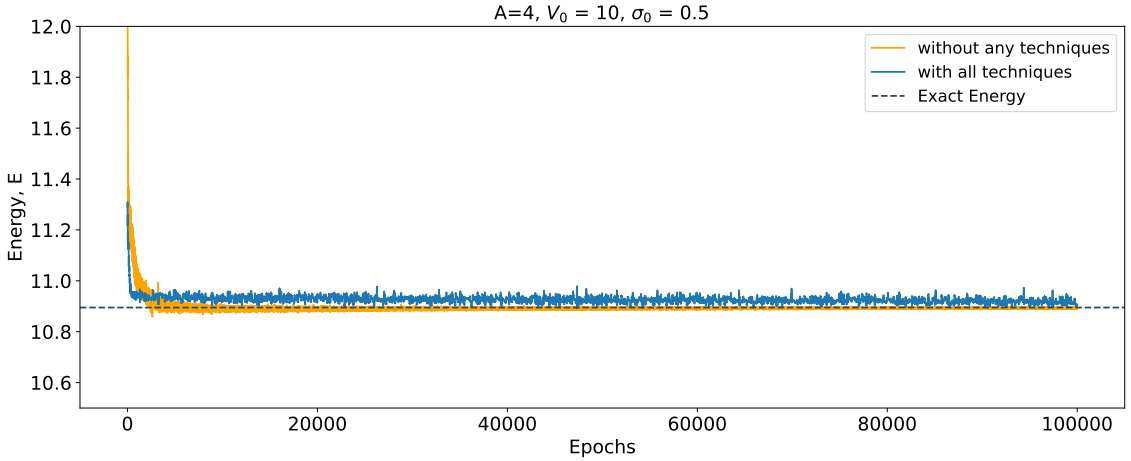


Figure 13: Energy minimization for two networks. The network represented by the orange line has none of the techniques studied applied to it, while the network represented by the blue line has all of them applied simultaneously (that includes Early Stopping, Transfer Learning, and the Scheduled Re-weighted MH sampling). For the blue line, we used 8000 walkers with 60 steps per walker. Finally, the dark dashed line represents the exact value of the energy computed by the exact diagonalization of the Hamiltonian.

Fig. 13 shows the results of optimizing two networks. The network represented by the blue line has Early Stopping, Particle and Interaction Transfer, and Re-Weighted MH applied to it. While the other network (orange line) has none of the studied techniques applied to it. The network with all the techniques reaches an energy of $E = 10.917 \pm 7 \times 10^{-3}$, while the one without any of the techniques reaches $E = 10.893 \pm 5 \times 10^{-3}$. It is true that the energy estimation is lower, but the model used to achieve that energy uses about 96.9% less training parameters and the training time is lowered by a factor of 10. It is important to note that the network with all the techniques has not converged yet as the Early Stopping was not triggered, so maybe with more training it could achieve a more accurate energy estimation.

5 Conclusions

In this project, we looked into studying a system of fully polarized, trapped, one-dimensional fermions with Gaussian interactions using a variation wavefunction in the form of an NQS. We introduce several methods to enhance the efficiency, accuracy, and scalability of these networks.

We looked into three different techniques specifically that can help improve the network's performance. The first is Early Stopping, which can be applied to increase the generalizability of the network and allow the training to stop once the network is considered converged.

The second technique is Transfer Learning. Within this framework, we looked into two different approaches. On the first hand, we looked into transferring the information of the network trained with an initial interaction $\{V_0', \sigma_0'\}$ into a new network that is trained with a target interaction $\{V_0, \sigma_0\}$. This approach provided faster convergence and lower variance in the energy calculations when applied during the training. On the other hand, we tried to transfer the information of a network trained with fewer particles into a new network that will be trained with more particles. Moreover, in this second approach we looked into freezing the equivariant layers to reduce the number of trainable parameters used in the network, which was proven to be beneficial as the network still managed to converge without re-training these parameters. Although this approach provided a substantial reduction in the number of variational parameters (up to 96.9% for the $A = 4$ case), it also led to decreasing the degrees of freedom the model has, resulting in additional noise being introduced to the network. Combining these techniques can allow us to study more complex interaction with larger number of particles. This can be done by starting with a small number of particles and relatively simple interaction, and then gradually approach the target interaction and number of particles by applying a combination of these two techniques.

The final technique studied in this project is the Scheduled Re-weighting of the MH sampling. In this method, it was shown that the sampling of the wavefunction does not need to happen at every epoch. Although this might produce spikes in the training, the network can manage to correct these errors and find the ground state energy. This approach provides a big improvement over the training time by reducing it by a factor of 10.

As we demonstrated, MH sampling takes a significant amount of time to generate samples. Therefore, a possible direction for future work is to explore alternative ways of sampling that could reduce the training time of the model. For instance, one could use more efficient proposal distributions, or employ faster MCMC methods (31; 32). Moreover, we can look into more efficient ways to remove the spikes presented in the training when the re-weighting techniques are applied, as increasing the number of walkers and steps per walker adds an additional temporal cost.

Other future lines of work include, studying different systems, with different type of interactions, or maybe polarized fermionic systems. Additionally, we can look into other starting points for Interaction Transfer, and how does the initial interaction impact the improvements provided by Interaction Transfer.

Bibliography

[1] G. Carleo and M. Troyer, "Solving the quantum many-body problem with artificial neural networks," *Science*, vol. 355, no. 6325, pp. 602–606, 2017.

[2] J. Carrasquilla, "Machine learning for quantum matter," *Advances in Physics: X*, vol. 5, no. 1, p. 1797528, 2020.

[3] O. Sharir, Y. Levine, N. Wies, G. Carleo, and A. Shashua, "Deep autoregressive models for the efficient variational simulation of many-body quantum systems," *Physical Review Letters*, vol. 124, no. 2, 2020.

[4] D. Pfau, J. S. Spencer, A. G. D. G. Matthews, and W. M. C. Foulkes, "Ab initio solution of the many-electron schrödinger equation with deep neural networks," *Phys. Rev. Res.*, vol. 2, p. 033429, Sep 2020.

[5] R. Zen, L. My, R. Tan, F. Hébert, M. Gattobigio, C. Miniatura, D. Poletti, and S. Bressan, "Transfer learning for scalability of neural-network quantum states," *Physical Review E*, vol. 101, no. 5, p. 053301, 2020.

[6] K. Choo, G. Carleo, N. Regnault, and T. Neupert, "Symmetries and many-body excitations with neural-network quantum states," *Physical Review Letters*, vol. 121, oct 2018.

[7] J. Keeble, M. Drissi, A. Rojo-Francàs, B. Juliá-Díaz, and A. Rios, "Machine learning one-dimensional spinless trapped fermionic systems with neural-network quantum states," *arXiv preprint arXiv:2304.04725*, 2023.

[8] L. Prechelt, "Early stopping-but when?," in *Neural Networks: Tricks of the trade*, pp. 55–69, Springer, 2002.

[9] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[10] F. Becca and S. Sorella, *Optimization of Variational Wave Functions*, p. 131–155. Cambridge University Press, 2017.

[11] P. Ring and P. Schuck, *The nuclear many-body problem*. Springer Science & Business Media, 2004.

[12] A. Rojo-Francàs, A. Polls, and B. Juliá-Díaz, "Static and dynamic properties of a few spin 1/2 interacting fermions trapped in a harmonic potential," *Mathematics*, vol. 8, no. 7, p. 1196, 2020.

[13] Y. Suzuki and K. Varga, *Stochastic variational approach to quantum-mechanical few-body problems*, vol. 54. Springer Science & Business Media, 1998.

[14] A. Krogh, "What are artificial neural networks?," *Nature biotechnology*, vol. 26, no. 2, pp. 195–197, 2008.

[15] C. C. Aggarwal, *Neural Networks and Deep Learning*. Springer, Sep 2018.

[16] A. Abraham, "Artificial neural networks," *Handbook of measuring system design*, 2005.

[17] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, R. Tibshirani, and J. Friedman, "Overview of supervised learning," *The elements of statistical learning: Data mining, inference, and prediction*, pp. 9–41, 2009.

[18] V. Verdhan and V. Verdhan, "Supervised learning for classification problems," *Supervised Learning with Python: Concepts and Practical Implementation Using Python*, pp. 117–190, 2020.

[19] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, R. Tibshirani, and J. Friedman, "Unsupervised learning," *The elements of statistical learning: Data mining, inference, and prediction*, pp. 485–585, 2009.

[20] T. Vieijra and J. Nys, "Many-body quantum states with exact conservation of non-abelian and lattice symmetries through variational Monte Carlo," *Physical Review B*, vol. 104, no. 4, p. 045123, 2021.
[21] H. Saito, "Method to solve quantum few-body problems with artificial neural networks," *Journal of the Physical Society of Japan*, vol. 87, p. 074002, jul 2018.
[22] K. Choo, A. Mezzacapo, and G. Carleo, "Fermionic neural-network states for ab-initio electronic structure," *Nature communications*, vol. 11, no. 1, p. 2368, 2020.
[23] C. P. Robert and G. Casella, *Metropolis–Hastings Algorithms*, pp. 167–197. New York, NY: Springer New York, 2010.
[24] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, "An introduction to MCMC for machine learning," *Machine learning*, vol. 50, pp. 5–43, 2003.
[25] Y.-Z. Zhang and M. D. Gould, "Quantum affine algebras and universal R-matrix with spectral parameter," *letters in mathematical physics*, vol. 31, pp. 101–110, 1994.
[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.
[27] J. W. Keeble, *Neural Network Solutions to the Fermionic Schrödinger Equation*. PhD thesis, University of Surrey, 2022.
[28] H. Noh, T. You, J. Mun, and B. Han, "Regularizing deep neural networks by noise: Its interpretation and optimization," *Advances in Neural Information Processing Systems*, vol. 30, 2017.
[29] A. Farahani, B. Pourshojae, K. Rasheed, and H. R. Arabnia, "A concise review of transfer learning," in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 344–351, IEEE, 2020.
[30] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.
[31] C. P. Robert, V. Elvira, N. Tawn, and C. Wu, "Accelerating MCMC algorithms," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 10, no. 5, p. e1435, 2018.
[32] E. Angelino, E. Kohler, A. Waterland, M. Seltzer, and R. P. Adams, "Accelerating MCMC via parallel predictive prefetching," 2014.
[33] P. Mehta *et al.*, "A high-bias, low-variance introduction to machine learning for physicists," *Physics reports*, vol. 810, pp. 1–124, 2019.

A Appendix A: Early Stopping Algorithm

In this appendix, we explain in more detail the algorithm used for the Early Stopping in Sec. 3.1. Algorithm 1 shows the early stopping process.

Algorithm 1 Early Stopping Algorithm

$triggers \leftarrow 0$
$errors \leftarrow 0$
$num\_epochs \leftarrow 0$
$i \leftarrow 0$
$S_{i-1} \leftarrow 0$
$E_{i-1} \leftarrow 0$
for epochs **do**
    $num\_epochs \leftarrow num\_epochs + 1$
    if $num\_epochs \geq window\_size$ **then**
        $S_i \leftarrow Calculate\_Slope(E_i)$
        $S_{diff} \leftarrow S_i - S_{i-1}$
        if $S_i \leq 3 \times 10^{-7}$ & $S_{i-1} \leq 3 \times 10^{-7}$ & $S_{diff} \leq 3 \times 10^{-6}$ **then**
            $E_{avg} \leftarrow avg(E_i) - avg(E_{i-1})$
            $V_{avg} \leftarrow avg(V_i)$
            $D_i \leftarrow std(E_i)$
            if $E_{avg} \leq 10^{-3}$ & $V_{avg} \leq 3 \times 10^{-3}$ & $D_i \leq 3 \times 10^{-3}$ **then**
                $triggers \leftarrow triggers + 1$
                if $triggers \geq 4$ **then**
                    $break$
            else
                $errors \leftarrow errors + 1$
        else
            $errors \leftarrow errors + 1$
        if $errors > 2$ **then**
            $triggers \leftarrow 0$
            $errors \leftarrow 0$
        $num\_epochs \leftarrow 0$
        $i \leftarrow i + 1$

In this algorithm, S_i denotes the Slope in window i, E_i denotes the energy points in window i, and V_i denotes the variance of the energy points in window i. Moreover, the $Calculate\_Slope$ function takes in the energy points at the window i and performs a linear fit of a line through this distribution of points. The function std calculates the standard deviation of the distribution of energy points at the window i. This is used to ensure that the points are relatively close together.

B Appendix B: Artificial Neural Networks

Artificial neural networks (ANN) (33), also known as neural networks (NN), are information processing systems modeled after the nervous system. This system is made up of numerous linked neurons that send information to one another. Similarly, ANNs simulate these networks so that a machine may learn and make decisions in a human-like manner. These networks form the basis for machine deep learning (DL) as they allow the machine to be able to extract the patterns from a training data set without human help.

ANNs are composed of a large number of interconnected nodes (representing human neurons) working together to solve a specific problem. ANNs can be trained to solve a specific problem like classifying, or pattern recognition. This training process is example-based, and the network is automatically modified by the machine to be able to solve this problem.

The explanation provided in this section is based on a book called "Neural Networks and Deep Learning: A Textbook" by Charu C. Aggarwal (15).

B.1 Artificial Neurons

The basic building block of an ANN is known as a node or neuron. These nodes receive an input signal then, after processing them, it produces an output signal as shown in figure 14. The inputs of a neuron are weighted, thus not all the input signals have the same influence. These input signals are combined together taking their weights into consideration,

$$\sum_{i=0}^{N} X_i W_i + b, \tag{21}$$

where N is the number of input signals, X_i is the input signal in position i, and W_i is the signal's corresponding weight. Finally, b is the learning bias of the node. The output of this summation is normally fed into an activation function that determines the neuron's output.

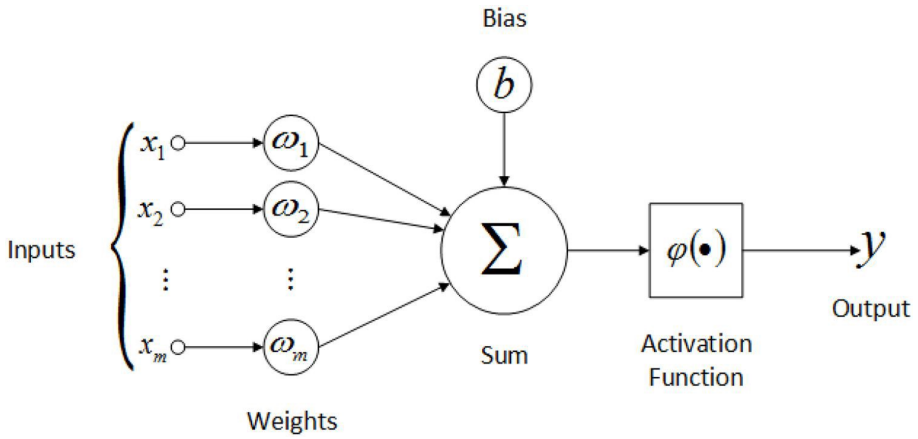


Figure 14: This figure shows the structure of an artificial neuron used in ANNs. This image is taken from a Medium post by Ricardo Mendoza (?)

B.2 Activation Functions

Activation functions take in the weighted sum of the input signal and determine the output of the neuron. There are many activation functions and each serves a specific purpose, and

choosing one depends on the problem that the network is trying to solve (15). The most basic activation used in neural networks is the linear activation,

$$\Phi(v) = v, \tag{22}$$

where Φ denotes the activation function, and v is the weighted sum of the neuron's inputs. This activation is mostly used when the output is expected to be a real number.

On the other hand, if the expected output is expected to have a value between $[0, 1]$, then a *Sigmoid* activation can be used. *Sigmoid* (shown in figure 15 image (c)) outputs values between 0 and 1, which is very helpful when the expected output is a probability. This activation is defined by the following equation:

$$\Phi(v) = \frac{1}{1 + e^{-v}}. \tag{23}$$

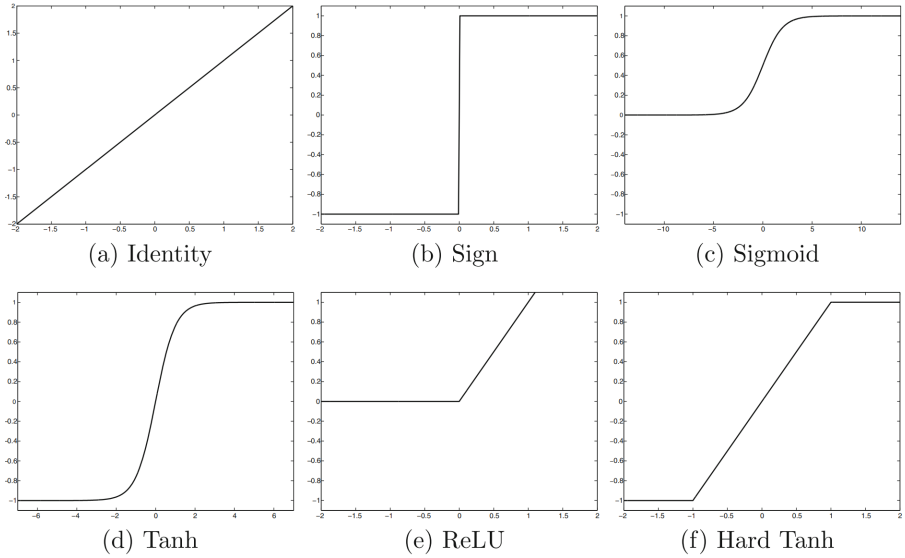
If the expected output is a binary value (0 or 1) then a *sign* activation can be used. This activation (shown in figure 15 image (b)) defines a cut-off at 0, where all the values below 0 are mapped to -1 and all values above 0 are mapped to one as defined in equation 24.

$$\Phi(v) = Sign(v) = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases} \tag{24}$$

Moreover, if the expected output is between $(-1, 1)$ then a tanh activation would be helpful. This activation (shown in figure 15 image (d)) is very useful when there is a desired distinguishability between positive and negative values in the output. This activation is given by the following equation:

$$\Phi(v) = \frac{e^{2v} - 1}{e^{2v} + 1}. \tag{25}$$

A modified version of this activation function known as Hard tanh (shown in figure 15 image (f)) has mostly replaced the soft tanh activation due to the ease it provides in training the network. Moreover, the Rectified Linear Unit (ReLU) (shown in figure 15 image (e)) has proven to be one of the most powerful activations in modern neural networks. This activation is known for its simplicity, speed, and faster convergence rates. The ReLU activation discards all the negative values by replacing them with a 0, while not changing the positive values.



(a) Identity (b) Sign (c) Sigmoid

(d) Tanh (e) ReLU (f) Hard Tanh

Figure 15: This figure shows the graphs of various activation functions. This image is take from the book "Neural Networks and Deep Learning: A Textbook" (15)

B.3 Loss Functions

The loss function measures the error in the output, which indicates the amount of change that needs to happen to the network's weights and biases. Similar to activation functions, the loss function is also problem dependent (15). For instance, for numeric outputs, a simple squared error function can be used that looks like $(y-\hat{y})^2$ for every training instance, where y is the expected output and $\hat{y}$ is the predicted output. Moreover, if the numerical output is between $[-1, 1]$ then a hinge loss can be used. This loss is defined by

$$Loss = max\{0, 1 - y\hat{y}\} \tag{26}$$

On the other hand, for probabilistic predictions, it depends on if the target is a binary prediction or a categorical prediction. If it is a logistic regression problem, then it would have a binary prediction and the following activation can be used:

$$Loss = \log(1 + \exp(-y\hat{y})) \tag{27}$$

If the output is a categorical probability, thus there are multiple output classes and each contains the probability of the output being that class. The loss function in this case is cross-entropy loss and it is given by

$$Loss = -\log(\hat{y}_r) \tag{28}$$

where r is the index of the correct class. There are many different types of activation and loss functions, however, choosing one depends on the other and on the nature of the output. Moreover, such a decision will change depending on the problem at hand as this will determine the nature of the output, the values this output can take, and the type of network to be used.

B.4 Multilayer Neural Networks

A single-layer neural network has an input layer where the data from the input is collected, and an output layer where the input information is processed and a decision is made. Note

that the input layer doesn't count as a layer in neural networks as no computation happens in this layer. However, this network is fairly simple as it only performs a weighted sum to the input data and applies an activation function to that. More complexity can be added to this network by adding more intermediate layers between the input and the output layer. The multi-layer neural network is known as a feed-forward network, as the output of every layer is used as the input to the consecutive layer. The user of such a network can only see the results of the output layer, all the intermediate calculations and results are hidden from the end user. This is why these layers are known as the hidden layers.

The architecture of a multi-layer network is defined by the number of hidden layers it has and the number of neurons each layer has. Another important factor to consider in the network design is the loss function to be used, and the activation functions used in each layer.

B.5 Training a Multilayer Neural Network

Training a neural network works by providing the network with a set of examples and then letting the network learn from the errors it makes. This process consists of two main steps. First, the network is given examples and it guesses the answer. Then, the error is calculated and the weights and biases across the network are modified accordingly. This process is repeated enough times until the network converges and the patterns in the input data are learned.

The first step in the learning process is known as the forward propagation step. In this step, the input is passed down the layers of the network. Each layer calculates the weighted sum of the input it was provided and then applies the activation function to that sum. The output of this operation is then passed on to the consecutive layer as the new input. Finally, when this reaches the output layer, the values produced by that layer represent the output of that network. This prediction produced by the output layer, is then compared to the expected result in the case of supervised learning. The error in this prediction is calculated according to the loss function defined in the network.

This error is then pushed back through the layers in a step known as back-propagation. One of the most popular tools used to calculate this back-propagation is gradient descent. In each layer, the gradient of the loss function with respect to the weights is calculated using the chain rule of derivatives. then this gradient is used to update the weights and biases in that layer. The gradient descent will help minimize the loss function, thus getting the network one step closer to convergence.

B.6 Practical Issues in Neural Network Training

The previously outlined training process does not always go smoothly. This could be due to a variety of factors, particularly two that are discussed in this section. The most common of which is overfitting. This problem occurs when a model produces good results when tested with the training dataset, but does not guarantee high performance when provided with new data. In other words, the model's training and testing data performance are vastly different. The most common cause of this issue is when the model is highly complex and there is very little data available to train it. As a result, instead of learning the patterns in the training data, the model memorizes them. There are several approaches to this problem, such as early stopping or adding noise to the training data, but the solution will ultimately rely on the network and the data available.

The other recurrent problem with networks with a very large depth is the vanishing and exploding gradient problem. This problem happens due to the chain rule applied when

calculating the gradient descent across the network in the back-propagation stage. In some cases, the gradient of the earlier layers becomes negligibly small by the time the updates reach the later layers. In other situations, however, these updates can become increasingly large, causing the learning process to be disrupted. In order to solve this issue, a change in the activation functions used in the hidden layers may help. Moreover, other approaches like adaptive learning or batch normalization can also be useful in this case.

B.7 Adam Optimizer

Gradient descent usually has a single constant learning rate that is used to update all the different weights. This is not very efficient as the learning rate plays a crucial role in the convergence of the network. If the learning rate is big, then the network most likely will never converge as the gradient descent will keep on fluctuating. On the other hand, if the learning rate is very small, then the network convergence process can be very slow. This can possibly cause some models not to converge because the gradient descent was not able to reach a minimum during the execution time.

In order to avoid this issue, adaptive learning rate methods are used. One of the most well known method is the adaptive moment estimation (Adam (26)) method is used. Adam overcomes this problem by maintaining a per-parameter learning rate that change based on the changes in the gradients.

The algorithm calculates the exponential averages of the recent gradients of a specific weight, along side the squared gradient. Moreover, the two parameters β_1 and β_2 are used to control the decay rates of these averages. These averages are estimates of the mean value, and the variance of the gradient. The parameters β_1 and β_2 are usually values between 0 and 1, where 1 is excluded. Normally, it is recommended that these values are initialized to 0.9.

The pseudo code for the algorithm as stated in the original paper can be seen in figure 16. As seen in this code, Adam calculates the first moment estimate m_t (representing the mean) and the second raw moment estimate v_t (representing the uncentered variance). These terms are computed as follows:

$$\begin{aligned} m_t &= \beta_1 \mathring{\text{u}} m_{t_1} + (1-\beta_1) \mathring{\text{u}} g_t \\ v_t &= \beta_2 \mathring{\text{u}} v_{t-1} + (1-\beta_2) \mathring{\text{u}} g_t^2 \end{aligned} \tag{29}$$

where t is the current timestamp, and g_t is the gradient at the current timestamp. Moreover, the g_t^2 indicates an element wise square ($g_t \odot g_t$).

Nevertheless, in case the parameters β_1 and β_2 are initialized as zero vectors, then the moment estimates are going to be bias towards zero in the first steps. This is why a bias correction step is done to ensure these effects are not present. This is done as follows:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{(1-\beta_1^t)} \\ \hat{v}_t &= \frac{v_t}{(1-\beta_2^t)} \end{aligned} \tag{30}$$

Finally the corrected moment estimates are used to calculate the changes to the weight (θ) as follows:

$$\theta_t = \theta_{t-1} - \alpha \mathring{\text{u}} \frac{\hat{m}_t}{\sqrt{\hat{v}_t}}, \tag{31}$$

where α is the learning rate.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 16: This figure shows the pseudo code for the Adam algorithm. This code is taken from the paper "ADAM: A Method For Stochastic Optimization" (26)