Solving Quantum Many-Body Problems with Neural Networks

Jane Kim
(Dated: October 30, 2019)

This project aims to solve for the ground state energy of a system of one-dimensional bosons using the variational Monte Carlo method enhanced by machine learning. The trial wave function is replaced by a feedforward neural network of one hidden layer. The natural cost function for a variational problem is the energy of the system, which we will compute stochastically and minimize with respect to the weights and biases of the network. The particular model of choice is exactly solvable, so the accuracy of the final result for the ground state energy and wave function can be measured quantitatively.

I. INTRODUCTION

The wave function Ψ of an isolated quantum system is a complex mathematical quantity that depends on its degrees of freedom and fully characterizes its state. For example, if the degrees of freedom are the spatial coordinates $\boldsymbol{x}$ of all the particles in the system, then the square of the wave function $|\Psi(\boldsymbol{x})|^2$ is the probability of the configuration occurring. The amount of information that needs to be encoded in the wave function grows exponentially with the number of particles in the system, posing a difficult computational problem for our limited classical resources. Typical methods that rely on calculating the wave function attempt to either compress the quantum state efficiently or sample a finite number of particle configurations that contribute the most to the description of the system. The goal of this project is to combine both approaches by using machine learning to enhance a traditional variational Monte Carlo calculation.

II. THEORY

A. Calogero-Sutherland Model

We will use this hybrid method to calculate the ground state energy of a simple 1D quantum system of N bosons. In the Calogero-Sutherland model, the bosons are trapped in a harmonic oscillator potential and interact via a pair-wise inverse squared potential. The Hamiltonian, or energy operator, of the system can be written as

$$\hat{H} = \sum_{i=1}^{N} \left(-\frac{1}{2}\frac{\partial^2}{\partial x_i^2} + \frac{1}{2}x_i^2 \right) + \sum_{i<j}^{N} \frac{\nu(\nu-1)}{(x_i - x_j)^2}, \qquad (1)$$

where x_i is the position of the ith particle, ν is an interaction parameter, and the constants $\hbar, m, \omega$ are set to one for convenience. Then the energy of the system in the state $\Psi(\boldsymbol{x})$ is given by

$$E = \frac{\int \Psi^\dagger(\boldsymbol{x})\hat{H}\Psi(\boldsymbol{x})d\boldsymbol{x}}{\int \Psi^\dagger(\boldsymbol{x})\Psi(\boldsymbol{x})d\boldsymbol{x}} \equiv \frac{\langle\Psi|\hat{H}|\Psi\rangle}{\langle\Psi|\Psi\rangle}. \qquad (2)$$

The most probable state for any isolated quantum system is the ground state. So it is of particular interest to calculate the ground state wave function Ψ_0 that produces the lowest possible energy E_0 using equation (2). For our system of bosons, these quantities can be found analytically and are given by

$$\Psi_0^{exact}(\boldsymbol{x}) = \exp\left(-\frac{1}{2}\sum_{i=1}^{N} x_i^2 \right) \prod_{i<j}^{N} |x_i - x_j|^\nu, \qquad (3)$$

and

$$E_0^{exact} = \frac{N}{2} + \frac{\nu}{2}N(N-1). \qquad (4)$$

Since this system is exactly solvable, we will be able to quantitatively measure the performance of our algorithm. Notice that in the non-interacting case, the bosons are only trapped in a harmonic oscillator, so the ground state is a simple gaussian

$$\Psi_0^{non\text{-}int}(\boldsymbol{x}) = \exp\left(-\frac{1}{2}\sum_{i=1}^{N} x_i^2 \right). \qquad (5)$$

B. Variational Monte Carlo

The variational principle states that for *any* guess for the ground state wave function Ψ, the expectation value of the energy is an upper bound for the true ground state energy:

$$E_0 \leq E = \frac{\langle\Psi|\hat{H}|\Psi\rangle}{\langle\Psi|\Psi\rangle}. \qquad (6)$$

This integral is typically very high dimensional, so it is calculated using Monte Carlo integration. For this principle to be useful, we need to provide a very good guess for the ground state wave function. The traditional approach would be to use physical intuition about the problem to design a parametrized trial wave function $\Psi_T(\boldsymbol{x};\boldsymbol{\alpha})$ that has expected limiting behaviors based on theory or a form inspired by experiment. By allowing the wave function to be "flexible", we can find a better upper bound for E_0 by minimizing E with respect to the parameters $\boldsymbol{\alpha}$.

One issue with this approach is that it heavily relies on the physical intuition of the researcher. If the trial wave function is parametrized in the wrong way, it will not provide a decent upper bound for the ground state energy. Another issue is that it lacks broad applicability, because each system encountered requires a new, physically motivated guess for the trial wave function.

C. Universal Approximation Theorem

What we need to overcome the limitations of the traditional variational Monte Carlo method is a maximally flexible trial wave function. Luckily, machine learning may provide the solution. The universal approximation theorem states that a feedforward neural network of just one hidden layer can approximate any continuous function, provided there are enough hidden neurons. This suggests we can use such a network to represent our trial wave function, and minimize the energy with respect to the weights and biases of the network. Not only could this approach alleviate the unnecessary restrictions on Ψ_T that are imposed by human intuition, but it could also allow us to repurpose the same trial wave function for a wide range of systems. In our case, the neural network will be designed to take continuous inputs and produce one real-valued output (this will be explained in more detail in the next section), so the same treatment could be applied to any continuous space model with a positive-definite ground state wave function.

III. METHOD

A. Neural Network Quantum State

We will represent the trial wave function of our system of bosons as a feedforward neural network with one hidden layer. The inputs are the positions of the N one-dimensional bosons and we can choose our network to have any number of M hidden neurons. We will notate the input layer and hidden layer as $\boldsymbol{x} \in \mathbb{R}^N$ and $\boldsymbol{h} \in \mathbb{R}^M$, respectively. They are fully-connected by weights $W \in \mathbb{R}^{M\times N}$ and are biased by $\boldsymbol{b} \in \mathbb{R}^M$, so that

$$\boldsymbol{h} = W\boldsymbol{x} + \boldsymbol{b}. \tag{7}$$

A general wave function is complex, but for a system of bosons, it is positive everywhere. This is convenient, because our network only needs to have one output neuron u, instead of two to represent the real and imaginary parts. The output of the network is given by

$$u = \boldsymbol{w}^T \boldsymbol{f}(\boldsymbol{h}), \tag{8}$$

where f is the activation function, which we will choose to be hyperbolic tangent, and the vector $\boldsymbol{w} \in \mathbb{R}^M$ contains the weights connecting the hidden neurons with the single output. We have bolded the activation function in (8) to emphasize that the result is a vector, with the function applied to each element of $\boldsymbol{h}$ separately. Finally, we define the trial wave function to be

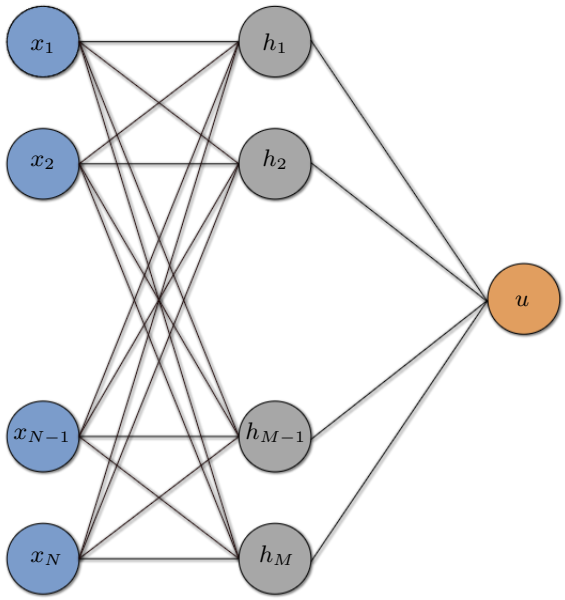

FIG. 1: Diagram of the feedforward neural network used to represent the trial wave function (9).

$$\Psi_T(\boldsymbol{x};\boldsymbol{\alpha}) = \exp(u) = \exp(\boldsymbol{w}^T \boldsymbol{f}(W\boldsymbol{x} + \boldsymbol{b})), \tag{9}$$

where we have collected all of the weights and biases of the network $(W, \boldsymbol{b}, \boldsymbol{w})$ into a single vector $\boldsymbol{\alpha} \in \mathbb{R}^{M(N+2)}$. A diagram of the network is shown in Figure 1.

B. Local Energy

To compute the energy in (2) efficiently, we will approximate the integral as a simple sum over local energies. First, notice that

$$E = \frac{\int \Psi_T^\dagger \hat{H} \Psi_T d\boldsymbol{x}}{\int \Psi_T^\dagger \Psi_T d\boldsymbol{x}} = \frac{\int |\Psi_T|^2 \frac{1}{\Psi_T} \hat{H} \Psi_T d\boldsymbol{x}}{\int |\Psi_T|^2 d\boldsymbol{x}}. \tag{10}$$

Since $|\Psi_T(\boldsymbol{x})|^2$ is the probability of the configuration $\boldsymbol{x}$ occurring under our assumptions about the form of the trial wave function, we can interpret the above expression as a weighted average of $\frac{1}{\Psi_T(\boldsymbol{x})} \hat{H} \Psi_T(\boldsymbol{x})$. Let the

local energy be defined by

$$E_L(\boldsymbol{x}) \equiv \frac{1}{\Psi_T(\boldsymbol{x})}\hat{H}\Psi_T(\boldsymbol{x}), \tag{11}$$

$$= \sum_{p=1}^{N}\left[\sum_{i=1}^{M} w_i W_{ip}^2 f''(h_i) - \frac{1}{2}\left(\boldsymbol{g}^T \boldsymbol{W}_p\right)^2\right] \tag{12}$$

$$+ \frac{1}{2}\sum_{p=1}^{N} x_p^2 + \sum_{p<q}\frac{\nu(\nu-1)}{(x_p - x_q)^2}, \tag{13}$$

where $\boldsymbol{g} \in \mathbb{R}^M$ is vector with elements

$$g_i = w_i f'(h_i), \tag{14}$$

and $\boldsymbol{W}_p$ is the pth column vector of the weight matrix W. Then we can approximate E as

$$E \approx \frac{1}{n}\sum_{i=1}^{n} E_L(\boldsymbol{x}_i) \equiv \langle E_L \rangle, \tag{15}$$

where n denotes the number of sampled configurations and $\boldsymbol{x}_i$ denotes the ith sample from the probability distribution $|\Psi_T(\boldsymbol{x})|^2$. For our model, $E_L(\boldsymbol{x})$ is the loss function, $\langle E_L \rangle$ is the cost function, and the goal is to minimize $\langle E_L \rangle$. The variance of the average local energy is

$$\sigma^2_{\langle E_L \rangle} = \langle E_L^2 \rangle - \langle E_L \rangle^2, \tag{16}$$

which can only equal zero if $\Psi_T = \Psi_0^{exact}$.

To minimize the average local energy $\langle E_L \rangle$, we will need to compute its gradient with respect to the parameters of the neural network $\boldsymbol{\alpha}$. It can be shown that

$$\nabla_{\boldsymbol{\alpha}}\langle E_L \rangle = 2\Big(\big\langle E_L \nabla_{\boldsymbol{\alpha}} \ln \Psi_T \big\rangle - \langle E_L \rangle \big\langle \nabla_{\boldsymbol{\alpha}} \ln \Psi_T \big\rangle\Big), \tag{17}$$

due to the hermiticity of the Hamiltonian $\hat{H}$ and the chain rule. We will need (11) and the analytical expression for $\nabla_{\boldsymbol{\alpha}} \ln \Psi_T(\boldsymbol{x}; \boldsymbol{\alpha})$ to compute (17). The latter can be found by taking derivatives of (9) with respect to the weights and biases $W, \boldsymbol{b}$, and $\boldsymbol{w}$, separately. The results are given by

$$\nabla_W \ln \Psi_T = \boldsymbol{g}\boldsymbol{x}^T, \tag{18}$$

$$\nabla_{\boldsymbol{b}} \ln \Psi_T = \boldsymbol{g}, \tag{19}$$

$$\nabla_{\boldsymbol{w}} \ln \Psi_T = \boldsymbol{f}(\vec{\boldsymbol{h}}). \tag{20}$$

C. MCMC Sampling

We will use Markov chain Monte Carlo (MCMC) sampling to pull particle configurations from the probability distribution $|\Psi_T(\boldsymbol{x})|^2$. The general approach is as follows:

1. Set the initial positions $\boldsymbol{x}_1$ of all N particles to random values from a Gaussian distribution.
2. For each iteration i, randomly pick one of the N particles and move it to a different position. Denote this trial configuration as $\boldsymbol{x}_T$.
3. Calculate the acceptance ratio $A(\boldsymbol{x}_T, \boldsymbol{x}_i)$ for the trial position $\boldsymbol{x}_T$ given our current position $\boldsymbol{x}_i$.
4. Generate a random number u between 0 and 1 from a uniform distribution. If $u \leq A(\boldsymbol{x}_T, \boldsymbol{x}_i)$, accept the move by setting $\boldsymbol{x}_{i+1} = \boldsymbol{x}_T$. Otherwise, reject the move by setting $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i$.
5. Repeat steps 2-4 until n samples have been obtained.

Since the positions of the particles are initially pulled from a Gaussian distribution, we should allow our sampler to reach equilibrium before performing any calculations with the samples. We can do this by simply ignoring the first 10% of samples.

There are many ways of implementing steps 2 and 3. In the brute force approach, we can define a new parameter s_{max} that determines the maximum step a particle can move in either direction. More explicitly, if we want to move particle p in the ith iteration, then the trial position of particle p is given by

$$x_T^{(p)} = x_i^{(p)} + s, \tag{21}$$

where s is a uniform random number between $-s_{max}$ and s_{max}. Then the acceptance ratio is the ratio of probabilities given by the trial wave function:

$$A(\boldsymbol{x}_T, \boldsymbol{x}_i) = \frac{|\Psi_T(\boldsymbol{x}_T)|^2}{|\Psi_T(\boldsymbol{x}_i)|^2}. \tag{22}$$

The brute force method is not ideal because it kicks particles to a different location completely randomly. A better approach would be to use what we know about Ψ_T to kick the particle into higher probability regions more often, maximizing the number of accepted moves. We can do this by using importance sampling, treating the particles as diffusive random walkers. The so-called quantum force on the pth particle is given by

$$F_p(\boldsymbol{x}) = 2\frac{1}{\Psi_T}\frac{\partial}{\partial x_k}\Psi_T = 2\boldsymbol{g}^T \boldsymbol{W}_k. \tag{23}$$

Then the Langevin and Fokker-Planck equations give a new trial position for the pth particle in the ith iteration according to

$$x_T^{(p)} = x_i^{(p)} + d\Delta t F_p(\boldsymbol{x}_i) + \xi\sqrt{\Delta t}, \tag{24}$$

where $d = 1/2$ is the diffusion constant, ξ is drawn from a normal distribution, and $\Delta t \in [0.001, 0.01]$ is a time step parameter. The transition probability is given by the Green's function

$$G(y, x) \propto \exp\left(-\frac{(y - x - d\Delta t F(x))^2}{4d\Delta t}\right), \tag{25}$$

so that the acceptance ratio for moving the pth particle is

$$A(\boldsymbol{x}_T, \boldsymbol{x}_i) = \frac{G(x_i^{(p)}, x_T^{(p)})|\Psi_T(\boldsymbol{x}_T)|^2}{G(x_T^{(p)}, x_i^{(p)})|\Psi_T(\boldsymbol{x}_i)|^2}, \tag{26}$$

$$= \exp\left(\frac{1}{2}(x_i^{(p)} - x_T^{(p)})\left(F_p(x_T^{(p)}) + F_p(x_i^{(p)})\right)\right. \tag{27}$$

$$\left. + \frac{1}{4}d\Delta t\left(F_p(x_i^{(p)})^2 - F_p(x_T^{(p)})^2\right)\right)\frac{|\Psi_T(\boldsymbol{x}_T)|^2}{|\Psi_T(\boldsymbol{x}_i)|^2}. \tag{28}$$

Notice that in both the brute force method and the importance sampling method, it is possible for the acceptance ratio to be larger than 1. When this happens, we say that the trial configuration $\boldsymbol{x}_T$ is more likely than the current position $\boldsymbol{x}_i$, so we should always accept the sample. However, this does not mean that we should always reject the sample when the trial configuration is less likely, otherwise the sampler would always converge to the most likely configuration and become static. Instead, we allow for the less likely configurations to be accepted with some corresponding probability.

D. Stochastic Gradient Descent

We will update the parameters of the neural network using stochastic gradient descent. In our case, stochasticity is introduced by replacing the true gradient of the energy integral with the gradient of the average local energy. This estimation is noisy, but much more computationally efficient. In contrast to stochastic gradient descent in the usual machine learning framework, in which only one data point is used to estimate the gradient, we will calculate the gradient (17) using the n samples of particle configurations at each iteration of the optimization process. Any configuration of particles is technically valid, although it may have an arbitrarily small chance of occurring. So our estimation could be viewed as taking a mini-batch of size n of the infinite number of the possible configurations.

Sometimes a fixed learning rate η can be used in gradient descent methods without much trouble. However, our estimations of the gradient are already noisy and have the potential to kick us out of the minimum we are trying to find. So it is important to use a learning rate that decreases towards zero as the optimization process progresses. For example, we can define the learning rate for the jth iteration of the process as

$$\eta_j = j^{-a}, \tag{29}$$

for some positive constant a. There is a lot of freedom here for the exact form of η_j, so we plan to experiment to find an option that converges quickly and reliably.

E. Algorithm

Many of the elements of a variational Monte Carlo calculation overlap with the elements of a typical machine learning algorithm. They both assume a set of random parameters, measure how wrong the result is based on the assumption, and have some way of changing the parameters in order to get a better result. The general outline of our hybrid algorithm is as follows:

1. Initialize the weights and biases of the neural network quantum state.
2. Sample n particle configurations from $|\Psi_T|^2$.
3. Estimate the local energy $\langle E_L\rangle$ and its gradient with respect to the network parameters $\nabla_{\boldsymbol{\alpha}}\langle E_L\rangle$ using the n samples.
4. Update the network parameters $\boldsymbol{\alpha}$ using stochastic gradient descent.
5. Repeat steps 2-4 until the magnitude of the gradient $\|\nabla_{\boldsymbol{\alpha}}\langle E_L\rangle\|$ is below some tolerance ϵ.

We do not want to allocate a large amount of memory to store all n particle configurations, so we will implement steps 2 and 3 concurrently. As each sample in the Markov chain is generated, each of their contributions to the average local energy and its gradient will be added to a running sum. These samples are used to update the parameters of the neural network, while the neural network is used to generate more samples. This feedback loop makes this a reinforcement learning problem.

Preliminary results indicate that the training ability of the network is very sensitive to the initial weights and biases. For some problems, it is sufficient to set these initial parameters to small random values near zero, but in our case, the gradient either blows up or vanishes. We will try to mitigate this issue by first training our network on the non-interacting case, the simple one-dimensional harmonic oscillator, for which the solution (5) is well-known. The trained network after this initial supervised learning of the weights will then be used as the initial state of our network for the reinforcement learning problem.

IV. IMPLEMENTATION

The method described in Section III will be implemented using five main classes corresponding to the five subsections A-E. The first is the class `FeedForwardNeuralNetwork` in the module `wavefunction`. An object from this class creates the network used to represent the trial wave function. It stores the number of inputs (or particles) N, the number of hidden nodes M, the weights and biases $\boldsymbol{\alpha} = (W, \boldsymbol{b}, \boldsymbol{w})$, and the current input $\boldsymbol{x}$. It includes functionalities to calculate the wave function (9), the local kinetic energy (12), the quantum force (23), and the gradient of $\ln\Psi_T$ with respect to the network parameters (18)-(20).

A `FeedForwardNeuralNetwork` object and the interaction parameter ν are used as inputs for the `CalogeroSutherland` class in the `hamiltonian` module. A `CalogeroSutherland` object can calculate the local energy (11), the exact ground state energy (4), and the exact ground state wave function (3). A `FeedForwardNeuralNetwork` object is also used to instantiate a `StochasticGradientDescent` object from the `optimizer` module. This object updates the parameters of the trial wave function given the gradient.

The `sampler` module has two classes: `BruteForce` and `ImportanceSampling`. They both take a `CalogeroSutherland` object and a sampling parameter (s_{max} or Δt) as inputs. In addition, they include methods for generating a new sample, calculating the appropriate acceptance ratio, and accepting trial samples.

Finally, the variational Monte Carlo method is implemented in the wrapper class `VariationalMonteCarlo` from the `vmc` module. It explicitly takes objects from the `hamiltonian`, `optimizer`, and `sampler` modules and implicitly takes an object from the `wavefunction` module as inputs. It computes the averages for the estimation of the local energy (11) and its gradient (17) and calls the optimizer to update the network parameters accordingly.

V. DATA

A. Visualization

Figures 2-4 show representations of the probability distributions given by the exact ground state wave function (3). The figures were generated by making a kernel density plot of all of the sampled positions of all the particles using the brute force sampling method. For an interaction parameter of $\nu = 1$, there are clear peaks corresponding to the most likely positions of the $N = 3, 5, 10$ particles. Of those peaks, the ones in the center are more likely than the others due to the harmonic oscillator potential. A kernel density plot was chosen because the wave function is easier to visualize with an arbitrary number of particles. The wave function of a system of bosons is also symmetric under particle exchange, so it makes sense to plot all the positions of the particles on the same axis. Similar plots will be made to show how the trial wave function converges to the exact ground state using the importance sampling method.

B. Preprocessing

The ground state wave function of a system of bosons must be symmetric under particle exchange, so we need to feed the data into the neural network in a form it can more easily detect and enforce the symmetry. For example, the configurations $\boldsymbol{x} = (-1, 0, 1)$ and $\boldsymbol{x} = (0, 1, -1)$ are equivalent for three bosons because they are indistinguishable particles. To ensure that these permutations of positions are treated the same, we will sort the sampled positions in ascending order before using them as input data.

VI. TRAINING

A. Parameters

It may be interesting to see how the network performs for different values of the interaction parameter ν. For our purposes, we will focus on the range $0 \leq \nu \leq 1$, with the end points corresponding to the one-dimensional harmonic oscillator problem. Notice that for $\nu = 1$ there is no interaction in the Hamiltonian but Figures 2-4 show some evidence of an interaction. More research will be required to explain this phenomena, but we are currently under the impression that $\nu = 1$ corresponds to changing the particles from bosons to fermions.

If the training of the network proves to be difficult, we may need to introduce the interaction potential more gradually. Assuming the changes in the network parameters are small for small changes in ν, we can first train the network for the non-interacting case then slowly increase ν to our desired value, optimizing the parameters as we go.

There are several additional hyperparameters we can tune to improve the performance of our algorithm. For example, we can explore how many hidden neurons M are required to well approximate the ground state, the learning rate η_j for stochastic gradient descent, and the parameters s_{max} and Δt used for sampling. We can also determine the best tolerance ϵ for stopping the optimization process, and the best number of samples n to accurately and efficiently approximate the energy and its gradient.

B. Learning Curve

For the supervised training of the neural network in the non-interacting case, we can compare Ψ_T with $\Psi_0^{non\text{-}int}$ by calculating the overlap integral

$$L \equiv \frac{\left[\int \Psi_T(\boldsymbol{x}) \Psi_0^{non\text{-}int}(\boldsymbol{x}) d\boldsymbol{x}\right]^2}{\int |\Psi_T(\boldsymbol{x})|^2 d\boldsymbol{x} \int |\Psi_0^{non\text{-}int}(\boldsymbol{x})|^2 d\boldsymbol{x}}. \tag{30}$$

We can estimate this integral using Monte Carlo

$$L \approx \frac{\langle \Psi_0^{non\text{-}int}/\Psi_T \rangle^2}{\langle \left(\Psi_0^{non\text{-}int}/\Psi_T\right)^2 \rangle}, \tag{31}$$

and train our network until L is sufficiently close to its maximum value of 1. We can do the same type of treatment for the reinforcement learning portion of the problem by calculating the overlap integral of Ψ_T and Ψ_0^{exact}.

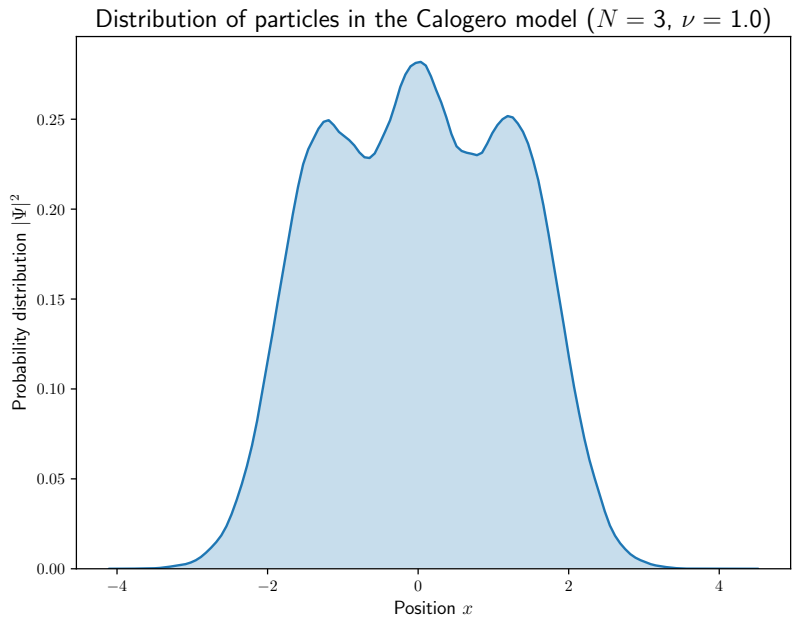

FIG. 2: Kernel density plot of 1000000 sampled positions of 3 particles from the exact distribution $|\Psi_0^{exact}|^2$ using the brute force sampling method.

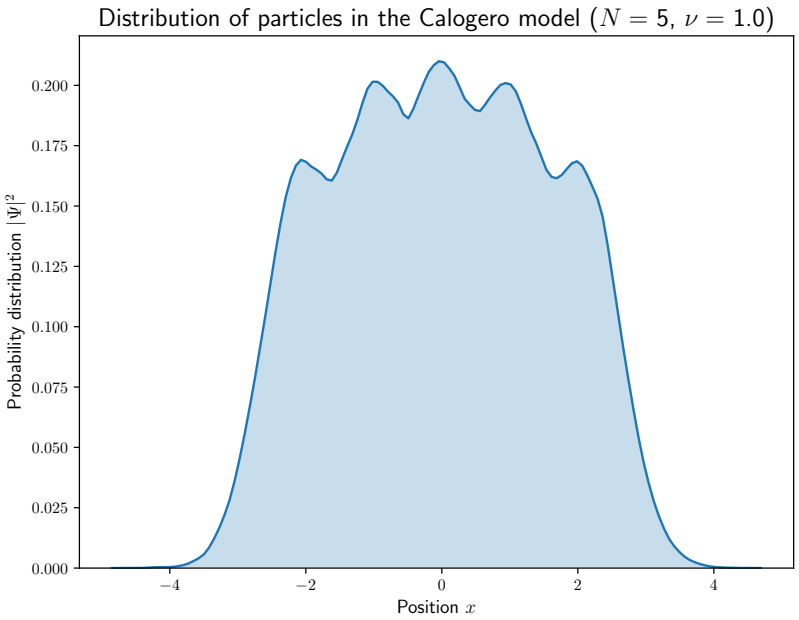

FIG. 3: Kernel density plot of 1000000 sampled positions of 5 particles using the brute force sampling method.

VII. POSTER

Since this neural network will be trained in an unconventional way, I plan on writing my code from scratch in Python. My poster will be structured similar to this proposal but contain much more concise information. It will include introduction, theory, method, implementation, data, and results sections. It will also include a visual representation of how the neural network converges to the exact solution. I intend on describing the structure of my code for the calculation in detail and dis-

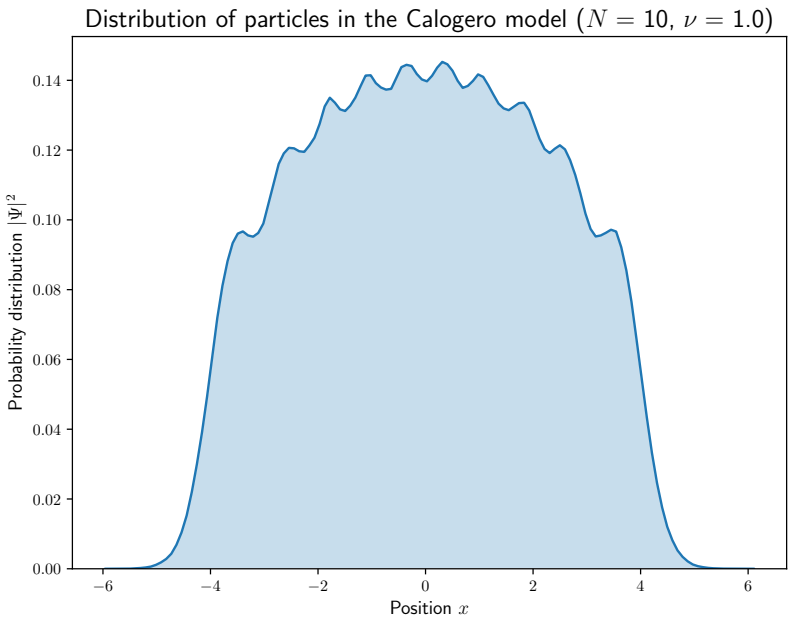

FIG. 4: Kernel density plot of 5000000 sampled positions of 10 particles using the brute force sampling method.

playing the relevant math in LaTeX. I will use Keynote to make the poster itself, and print it at the MSU Main Library. For maximum efficiency, the training will be performed on the HPCC cluster at ICER. The poster will be held onto the wall by tape or magnets.

VIII. TIMELINE

- November 3 - Implement the supervised training of the initial weights and calculation of the overlap integrals.
- November 10 - Implement the sorting of positions and perform parameter tuning.
- November 17 - Create a `Plotter` class that can run the hybrid algorithm and generate various plots.
- November 24 - Start poster and generate plots to show the evolution of the trial wave function.
- December 1 - Finalize and print poster.