Solving Quantum Many-Body Problems with Neural Networks

By Jane Kim

Introduction

This project aims to solve for the ground state energy of N interacting bosons trapped in a one-dimensional harmonic oscillator. Typical approaches to this type of problem include variational methods in which a parametrized wave function is designed based on physical intuition, used to estimate the energy via Monte Carlo integration, and tuned to return the lowest energy possible. However, the dependence of the trial wave function on human insight introduces significant bias and restricts the ability to study unknown systems. Here, we replace the trial wave function with a feedforward neural network with only one hidden layer. The inputs are the coordinates of the bosons which are sampled using the outputs, making this a reinforcement learning problem. The cost function is the energy of the system, which is computed stochastically and minimized with respect to the weights and biases of the network.

Theory

The Hamiltonian for the Calogero-Sutherland model of one-dimensional bosons is given by

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

where x_i is the position of the ith particle and ν is an interaction parameter. We will approximate the energy of the system in an arbitrary state $\Psi(\vec{x})$ as an average over sampled local energies $E_L = \frac{1}{\Psi}\hat{H}\Psi$:

$$E = \frac{\langle \Psi | \hat{H} | \Psi \rangle}{\langle \Psi | \Psi \rangle} \approx \frac{1}{n}\sum_{i=1}^{n} E_L(\vec{x}_i) \equiv \langle E_L \rangle,$$

where $\vec{x}_i$ is the ith particle configuration sampled from the distribution $|\Psi(\vec{x})|^2$ and n is the number of samples. The variational principle states that the energy of a system in any state Ψ is an upper bound for the true ground state energy E_0. For the Calogero-Sutherland model, there exists an exact solution for this ground state energy and its corresponding ground state wave function:

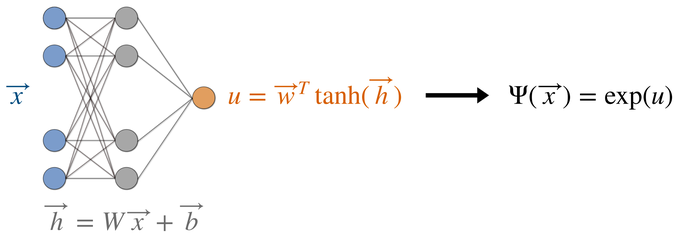
$$\Psi_0^{exact}(\vec{x}) = \exp\left(-\frac{1}{2}\sum_{p=1}^{N} x_p^2 \right) \prod_{p<q} |x_p - x_q|^{\nu},$$

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

These known solutions will be used to quantitatively measure the accuracy of the final result.

Model

The wave function of a system of bosons is positive-definite so the trial wave function can be replaced by the exponentiated output of a feedforward neural network:



In general, the accuracy improves as number of hidden units M increases, but the gain is relatively minimal after $M \simeq 20$. To help the network learn the particle exchange symmetry of the wave function, the inputs $\vec{x}$ will be sorted in ascending order. Importance sampling will be used to sample random particle positions consistent with the probability distribution $|\Psi|^2$ and the Adam algorithm will be used to optimize the weights W, $\vec{w}$ and biases $\vec{b}$. These network parameters will be collectively referred to as $\vec{\alpha}$.

Implementation

Proper initialization of the weights and biases of the neural network is crucial. Since the particle positions are sampled from the trial wave function given by the neural network, vanishing or exploding gradients quickly emerge because the sampling algorithm "kicks" a particle far from the origin very early on in the calculation. So at the very least, the initial weights and biases should be chosen so that the initial wave function Ψ is localized about the origin. We accomplish this by first training the neural network in a supervised setting on the wave function for the non-interacting case $\nu = 0$, i.e. a gaussian distribution.

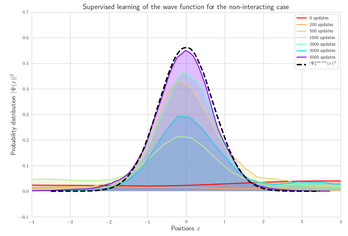
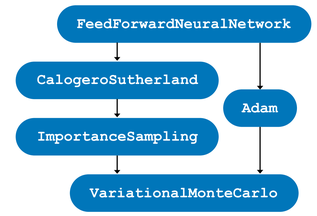


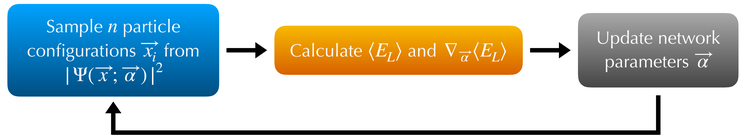
Figure 1: Kernel density estimation of the spatial distribution of two particles during the supervised learning of the wave function for the non-interacting case. The number of hidden units is $M = 20$ and the number of samples is $n = 300000$.
I will make the font on this plot bigger

After the non-interacting case is learned, abruptly introducing the singularity in the interaction potential also causes the gradient to explode. To limit this effect, we gradually increase the strength of the interaction potential at a rate γ and train the network along the way.

Code Structure



This model is implemented using five main classes that feed into one another. The structure is shown in the diagram on the right, where each class requires information from the class(es) above it. All that is needed in the main program is to instantiate each object with its corresponding input parameters and call the `minimize_energy()` function belonging to the `VariationalMonteCarlo` class. The basic algorithm is as follows:



Hyperparameters

I will make a table of all the relevant hyperparameters and the values I used

Results

Here I will have a plot of my best attempt at the reinforcement learning part of the problem and a comparison with the exact solution