Project 2 - An Autoregressive Model for Bosonic Quantum Dots

Erlend Lima, Kristian Wold

A novel neural network architecture has been built to obtain uncorrelated samples from the position wave function in finite potentials. In this paper, we study bosons in a harmonic oscillator in one, two and three dimensions, both in the interacting and non-interacting case. In the interacting case, a shielded Coulomb potential on the form

$$V(\vec{r_1}, \vec{r_2}) = \frac{\alpha}{\sqrt{|\vec{r_1} - \vec{r_2}|^2 + \beta^2}}$$

was used as interaction. For two and six non-interacting bosons, the ground state energy was calculated to a precision of about 0.1% with respect to the analytical result for all number of dimensions . For two interacting bosons, the ground state energy was calculated to a precision of about 1% with respect to the DMC and CISD calculations used as benchmarks for all number of dimensions.

The one body densities were estimated to some accuracy, but with some notable flaws. In the 1D case for two interacting bosons, the one-body density displayed the characteristic seperation of two peaks, although not as sharply as the benchmark. The one body density also suffered from a slight asymmetry. In higher dimensions, the radial one body density varied slightly from the benchmark, for both the interacting and non-interacting case.

I. INTRODUCTION

Problems in many-particle quantum mechanics is notoriously hard, and has over many years resulted in a suit of methods and techniques for solving them efficiently and reliably. A popular family of such methods is Variational Monte-Carlo, which parameterize a trial wave function and seeks to optimize the parameters in order to minimize the expected energy, thus approximating the ground state of the system. While classic VMC methods, such as trial wave functions with Jastrow factors, have proved useful, they come with some weaknesses: the inflexibility of the trial wave function may lead to poor approximations of the real ground state, even if a global minimum in energy for the particular trial wave function was obtain. Further, observables are hard to estimate from trial wave functions, because popular methods for sampling, such as Metropolis-Hastings, tends to produce data with high degree of auto-correlation[1]. This is even more true for larger systems, and leads to a high statistical uncertainty in estimated observables.

Drawing inspiration from Deep Learning, Artificial Neural Networks (ANN) have in recent years been employed to model wave functions with great success[2]. Because of the huge number of parameters, ANNs tend to be much more flexible than classical trial wave functions. However, there is the new challenge to optimize the parameters, and the usual problem with auto-correlated samples is still present.

To address the latter problem, recent advances[3, 4] have succeeded in employing auto-regressive models on spin-systems, such as Quantum Ising-model. To obtain a spin-configuration, each spin is sampled in sequence, each time conditioning on the previous sampled spins. Because of the nature of auto-regressive models, all sampled spin-configurations are uncorrelated, removing the problem with high statistical variance in estimating observables.

In this paper, we present an autoregressive model for bosonic quantum dots based on Recurrent Neural Networks, inspired by the work of Hibat-Allah Et al. [3].

II. THEORY

A. The System

1. *The Potentials*

Throughout this paper, the usual natural units $m = \hbar = 1$ are used. The Hamiltonian under investigation describes N bosons in a potential trap of the form

$$H = \sum_{i=1}^{N}\left(\frac{-\hbar^2}{2m}\nabla_i^2 + V_{\text{ext}}(\mathbf{r}_i)\right) + \sum_{i<j}^{N} V_{\text{int}}(\mathbf{r}_i, \mathbf{r}_j)$$

where V_{ext} is the external potential of the trap and V_{int} is the internal potential between the particles. The external potential is a symmetric, harmonic oscialltor potential in three dimensions:

$$V_{\text{ext}}(\mathbf{r}) = \frac{1}{2}\omega\left(x^2 + y^2 + z^2\right). \tag{II.1}$$

The corresponding 2D and 1D harmonic potential is gotten by removing the z and y dependancy from the above expression, in that order.
The internal potential is a shielded Coloumb potential:

$$V(\vec{r_1}, \vec{r_2}) = \frac{\alpha}{\sqrt{|\vec{r_1} - \vec{r_2}|^2 + \beta^2}} \tag{II.2}$$

In the case that $\beta = 0$, the potential reduces to normal Coloumb interaction, and in the case $\alpha = 0$, the system becomes non-interacting.

B. Variational Monte Carlo

In order to find a good candidate wavefunction for a given potential, one can employ the *variational principle*. One starts by guessing a trial wavefunction $|\Psi_T\rangle$ and estimating the trial energy, which is guaranteed to be equal to or higher than the true ground state energy E_0:

$$E_0 \leq \langle E\rangle = \frac{\langle\Psi_T|H|\Psi_T\rangle}{\langle\Psi_T|\Psi_T\rangle}. \tag{II.3}$$

If $|\Psi_T\rangle$ is an eigenfunction of the Hamiltonian, the variance σ^2 will be minimal

$$\sigma^2 = \frac{\langle\Psi_T|H^2|\Psi_T\rangle}{\langle\Psi_T|\Psi_T\rangle} - \left(\frac{\langle\Psi_T|H|\Psi_T\rangle}{\langle\Psi_T|\Psi_T\rangle}\right)^2 = 0.$$

The variational principle expands on this idea by letting $|\Psi_T\rangle$ be a functional class of the θ, which is an arbitrary finite set of parameters. By varying the parameters θ one can find the optimal trial wavefunction within the functional class by minimizing $\langle E\rangle$.
Only a small collection of potentials have analytical solution using the variational principle. For most potentials, one must numerically integrate (II.3) using Monte Carlo integration.

For a stochastic variable x with probability density function $p(x)$, the average $\langle x\rangle$ is defined as

$$\langle x\rangle = \int_{\mathbb{R}} xp(x)\mathrm{d}x.$$

By sampling the stochastic variable M times, the average can be approximated by

$$\langle x\rangle = \int_{\mathbb{R}} xp(x)\mathrm{d}x \approx \frac{1}{M}\sum_{i=1}^{M} x_i p(x_i).$$

Applying this to an observable $\mathcal{O}$, we have

$$\begin{aligned}\langle\mathcal{O}\rangle &= \langle\Psi|\mathcal{O}|\Psi\rangle\\ &= \int \mathrm{d}\mathbf{r}\Psi^*\mathcal{O}\Psi\\ &= \int \mathrm{d}\mathbf{r}|\Psi|^2\frac{1}{\Psi}\mathcal{O}\Psi\\ &= \frac{1}{M}\sum_{i=1}^{M} p(\mathbf{r})\mathcal{O}_L.\end{aligned}$$

where $|\Psi|^2$ is defined as the probability density function, and $\frac{1}{\Psi}\mathcal{O}\Psi$ the *local operator*.
The local trial energy can then be defined as

$$E_L = \frac{1}{\Psi_T}H\Psi_T$$

which can be computed using Monte Carlo integration as

$$\langle E_L\rangle \approx \frac{1}{M}\sum_{i=1}^{M} p(\mathbf{r}_i)E_L(\mathbf{r}_i).$$

One way of optimizing the parameters θ is by gradient descent, for which we need the gradient $\frac{\partial}{\partial\theta}\langle E\rangle$. It can be shown that for a trial wave function ψ, the gradient is given by

$$\frac{\partial}{\partial\theta}\langle E\rangle = 2\left\langle\frac{\psi'}{\psi}(E_L - \langle E_L\rangle)\right\rangle = 2\left\langle\log(\psi)'(E_L - \langle E_L\rangle)\right\rangle,$$

where the derivative is with respect to θ.

C. Onebody Density

A useful way to understand a many body system is to integrate over all dimensions except for one, yielding the *one body density* $\rho(\mathbf{r})$, defined as

$$\rho(\mathbf{r}) = \int \mathrm{d}\mathbf{r}_2 \ldots \mathrm{d}\mathbf{r}_N \left|\Psi_T(\mathbf{r})\right|^2.$$

It is a scaled probability density function giving the number of particles within the volume $\Delta\mathbf{r}$ as $\rho(\mathbf{r})\Delta\mathbf{r}$. By convention the integral over all d$\mathbf{r}$ yields the total number of particles in the system, N.

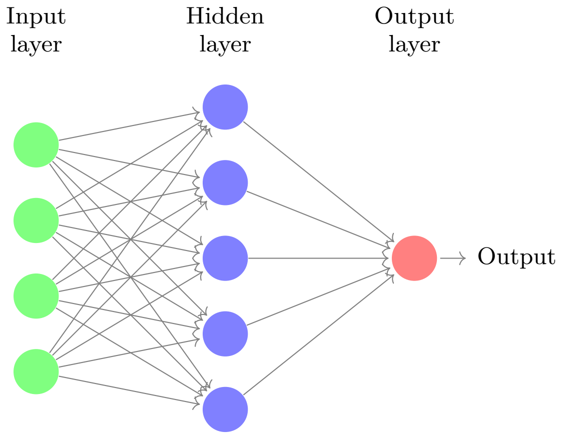

Figure II.1: Diagram of an artificial neural network with one fully connected hidden layer. Taken from http://www.texample.net/tikz/examples/neural-network/

D. Metropolis-Hastings

The estimate of the local energy relies on samples from the trial wave function. To get a physical value, the configuration of the particles must be as physical and probable. As the configuration achieving this is unknown, the configuration space must be explore. This is done through Monte Carlo simulation, more specifically by using the Metropolis-Hastings algorithm.
At each MC step a single particle is chosen at random, and a change to its position is proposed by moving it a fixed step length Δ and computing the ratio between new and old probability densities

$$\omega = \frac{P\left(\mathbf{r}_r, \ldots, \mathbf{r}_k^*, \ldots, \mathbf{r}_n\right)}{P\left(\mathbf{r}_r, \ldots, \mathbf{r}_k, \ldots, \mathbf{r}_n\right)} = \frac{\left|\Psi_T\left(\mathbf{r}_r, \ldots, \mathbf{r}_k^*, \ldots, \mathbf{r}_n\right)\right|^2}{\left|\Psi_T\left(\mathbf{r}_r, \ldots, \mathbf{r}_k, \ldots, \mathbf{r}_n\right)\right|^2}$$

where $\mathbf{r}^*$ denotes a modified position. If the ratio ω is greater than a uniformly distributed number $\theta \in [0, 1]$, the move is accepted. This ratio can often be reduced analytically to obviate the need for recomputing the entire probability density each step.

E. Neural Networks

Neural networks is a machine learning algorithm inspired by biological neurons, and has achieved impressive performance on a wide class of problems. A neuron is modeled by a *perceptron*, with perceptrons stacked together in layers and layers stacked together into a network architecture. The first layer takes an input vector $\mathbf{x}$ of d features and produces an *activation* $a_i(\mathbf{x})$. The activation is feed into the next layer, called *hidden layer*, and so on until the last layer, the *output layer*.
A layer transforms its input but first applying an affine transformation on the form

$$z^{(i)} = W^{(i)}\mathbf{x} + b^i$$

with W^i being the *weights* of layer i and b^i its *bias*. This is followed by a non-linear transformation

$$a_i(\mathbf{x}) = f_i(z^{(i)})$$

The *activation function* f_i is specific to each layer. The activation function of the output layer determines whether the network is a classifier, in which the function is a sigmoid or softmax function, or regression, where the function are unbounded linear functions.
The combination of linear and non-linear transformations allows a network to approximate any function to arbitrary precision, a result known as the universal approximation theorem[5]. In order to achieve this expressive power, the network has to be trained. This is done by an algorithm known as *backpropagation*.

1. *Backpropagation*

To train a network, a cost function is optimized, often being the mean square error. The optimization is done by gradient descent, where an input is fed forward through the network, compared to the expected output, and the error is fed backwards and used to update the weights and biases. This backpropagation of the error is essentially the chain rule from calculus applied recursively.
The four equations of backpropagation are

$$\begin{aligned}
\delta_j^l &= \frac{\partial E}{\partial a_j^l} f\prime\left(z_j^l\right) \\
\delta_j^l &= \frac{\partial E}{\partial b_j^l} \\
\delta_j^l &= \left(\sum_k \delta_k^{l+1} w_{kj}^{l+1}\right) f\prime\left(z_j^l\right) \\
\frac{\partial E}{\partial w_{jk}^l} &= \delta_j^l a_k^{l-1}
\end{aligned}$$

A deeper explanation of these equation and their derivation can be found in literature, such as [5].

F. Recurrent Neural Networks

A recurrent neural network (RNN) is an extension to vanilla neural networks. Its purpose is to capture the current context through a hidden state whereby long range correlations can be modeled. Its usefulness is

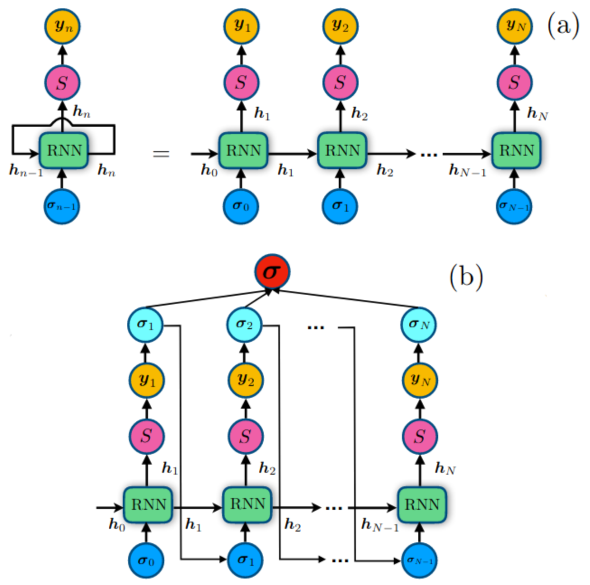

Figure II.2: Illustrations of common recurrent neural net. (a) Left: A RNN cell takes a sequence of inputs $\{\sigma\}$ that are combined with the hidden state $\mathbf{h}_{n-1}$ to produce an updated hidden state $\mathbf{h}_n$, encoding information about σ_{n-1}. The hidden state is here propagated through a softmax function S to produce the output $\mathbf{y}_n$, but note that a softmax is not required. Right: The unrolled version, more clearly showing the sequential nature of RNN. (b) A representation of autoregressive sampling of RNNs. Instead of taking external sequential input, the RNN itself generates the sequence $\{\sigma\}$ from an initial state σ_0. The output of each propagation through the RNN is fed back into the network, with their combination being the ultimate output. Edited version of figure from [3].

apparent when there are strong correlations between sequential inputs, such as time varying signals.
A RNN can model a correlated probability distribution where a new sample is entirely determined by previous samples. Letting $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ denote a configuration of N samples, the probability of observing a specific configuration is

$$P(\boldsymbol{\sigma}) = P(\sigma_1)P(\sigma_2|\sigma_1)\cdots P(\sigma_N|\sigma_{N-1}, \ldots, \sigma_1)$$

This is achieved through the use of a *recurrent cell.* The basic structure of a recurrent cell is and update of the hidden state by combining it with the input, and using the updated state to compute the output. The hidden state $\mathbf{h}_{n-1}$ contains information about the previous forward pass, and is updated by concatenating the hidden state to an input state $\boldsymbol{\sigma}_{n-1}$ and passing it through a non-linear activation function f such that

$$\mathbf{h}_n = f(W\,[\mathbf{h}_{n-1}; \boldsymbol{\sigma}_{n-1}] + \mathbf{b})$$

with $\mathbf{h}_{n-1}, \mathbf{b} \in \mathbb{R}^{d_h}$, $\boldsymbol{\sigma}_{n-1} \in \mathbb{R}^{d_v}$ and $W \in \mathbb{R}^{d_h \times (d_h + d_v)}$ where $d_{h,v}$ are the dimensions. The activation function is often tanh due to its desirable properties.

The updated hidden state is propagated forward through a softmax layer S to yield the coefficient of each σ_n:

$$\mathbf{y}_n = S(U\mathbf{h}_n + \mathbf{c})$$

where $U \in \mathbb{R}^{d_v \times d_h}$ and $\mathbf{c} \in \mathbb{R}^{d_v}$.
The softmax function ensures that the coefficients $\mathbf{y}_n$ sum up to 1, forming a probability distribution over the states σ_n. This allows for the full probability to be computed as

$$P(\boldsymbol{\sigma}) = \prod_{n=1}^{N} \mathbf{y}_n \cdot \sigma_n.$$

To sample N samples from the RNN probability distribution, one begins with initial states $\boldsymbol{\sigma}_0$ and $\mathbf{h}_0$, computing the resulting σ_1, one-hot encoding it to obtain σ_1, repeating the computation with σ_1 and the now updated $\mathbf{h}_1$, and iterating until N samples are obtained.

G. RNN Wave functions

In general, wave functions are complex valued. Our problem domain, however, only concerns itself with stochastic many-body Hamiltonian at ground state energy. These yield strictly real and positive wave functions[4]. For a many-body system of N particles, the total wave function in position basis can be written as the product of all conditional single particle wave functions

$$\Psi(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_N) = \prod_{i=0}^{N} \psi_i(\mathbf{x}_i|\mathbf{x}_{i-1}, \ldots, \mathbf{x}_0)$$

for positions $\{\mathbf{x}_i\}$. That each ψ_i is normalized is a sufficient condition for the total wave function to be normalized[4]. However, this was not done here, as the wave function itself is only used as a ratio in the Metropolis sampling, eliminating the need for normalizing. The conditional single particle wave functions can be obtained through the use of autoregressive RNN, illustrated in (b) of fig. II.2, with $\boldsymbol{\sigma}_i = \mathbf{x}_i$.

III. METHOD

A. Architecture

A novel neural network architecture has been built to obtain uncorrelated samples from the position wave function of bosons in finite potentials. A diagram is shown in fig. III.1. It consists of three main parts: a DNN, a RNN and a MCMC sampler.

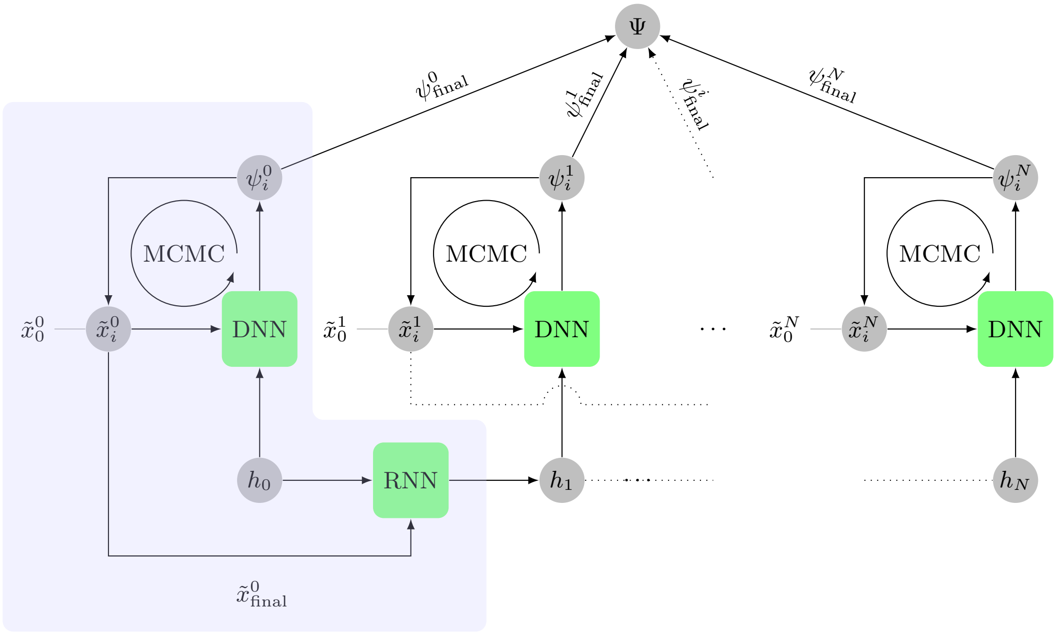

Figure III.1: Diagrammatic overview of the neural network architecture for a system of N bosons. The shaded blue region is a single "cell" consisting of a RNN to update the hidden state, a DNN to evaluate the position wave function, and a brute force Metropolis sampler to sample the position space. The Metropolis sampling yields a probable position for the particle, $\tilde{x}^i_{\text{final}}$, which is taken as the "actual" position of the particle and used in the RNN to update the hidden state. Each particle in the system corresponds to one cell, with the subsequent particles depending on the former through the hidden state. Once all of the particles have been iterated through, the combined wave function Ψ can be obtained from their single particle wave functions. Each MCMC sampling is initialized by random positions $\tilde{x}^i_0$

The DNN receives the position $\mathbf{x}_i$ of a particle concatenated with the hidden state from the RNN as input and evaluates the conditional wave function $\psi(\mathbf{x}_i|\mathbf{x}_{i-1},\cdots,\mathbf{x}_0)$. It has two fully connected layers of 32 or 64 neurons. Both tanh and *ReLu* were tried as activation functions to study their influence on the result.

The position space is explored by the MCMC sampler to obtain the most probable positions. A brute force Metropolis sampler is used for this purpose. The sampling is initialized by a number of walkers with random position drawn from a uniform distribution $U(\mathbf{x}_{\min},\mathbf{x}_{\max})$, where the bounds were set to ensure that most ($> 97\%$) of the probability distribution is enclosed. After a set number of iterations the sampler returns the last sampled position, which is used as the *actual* position of the particle in question.

The RNN takes the last sampled position to update the hidden state. It has a single layer with five neurons and a tanh activation function. By encoding the position of the last particle in the hidden state, the wave function of the subsequent particle becomes dependent on the position of all previous particles by the autoregressive property of the RNN. The resulting Metropolis samples are therefore uncorrelated, obviating the need for blocking.

B. Automatic Differentiation in Tensorflow

Since we are studying Hamiltonians defined on continuous space, calculating the Laplacian of the trial wave function is necessary to account for the kinetic energy of the system. Since the RNN-DNN model is made using the popular neural network framework Tensorflow(2), it feels natural to use its automatic differentiation functionality to calculate the Laplacian of the model with respect to the inputs(the positions). However, for more than one dimensional input, it turns out to be quite difficult, although a work-around proposed by [6] resolved the difficulty. The problem was as follow:
Starting with an input x of dimension $(\mathsf{batch\_size},\ \mathsf{d})$, a forward pass through the model results in an output y of dimension $(\mathsf{batch\_size}, 1)$, as the wave function evaluates to a scalar value. When performing `grad_y = tf.gradients(y,x)`, we get

$$\mathsf{grad\_y} = \left(\frac{\partial y}{\partial x_1},\cdots,\frac{\partial y}{\partial x_d}\right),$$

which has dimensional $(\mathsf{batch\_size}, d)$. The problem arises when calculating the second derivative `grad2_y = tf.gradients(grad_y,x)`, symbolically

$$\mathsf{grad2\_y} = \left(\sum_{i=1}^{d}\frac{\partial^2 y}{\partial x_1\partial x_i},\cdots,\sum_{i=1}^{d}\frac{\partial^2 y}{\partial x_1\partial x_i}\right).$$

Since `tf.gradients` is meant to be used for gradient descent, a summation of the derivatives of all outputs w.r.t the input x is performed, which should not be present in the Laplacian. The naïve solution is to slice the output after the first differentiation, such as `tf.gradients(grad_y[i],x[i])`. Sadly, slicing this way does not respect the computational graph constructed during forward pass, and `x[i]` is interpreted as a new variable not related to `grad_y[i]`. The fix is to use clever reshaping of the input variable `x` before passing it to the model, in order to allow for the wanted slicing without destroying the computational graph. Details can be found in [6] and in our source code.

C. Optimization

During the training procedure of the RNN-DNN model, we use the following expression for the loss value:

$$L = 2\sum_{i=1}^{N_B} \ln(\psi_i)(E_i - E)$$

where N_B is the batch size, ψ_i and E_i are the wave function and local energy of sample i of the batch, and E is the average local energy over the whole batch. During gradient decent, the derivative of the loss value is differentiated w.r.t. the model parameters, and the gradient describes (ref) is recovered.

During vanilla gradient decent, the parameters are updated by performing a small step in the opposite direction of the computed gradient

$$\theta \rightarrow \theta - \mu\nabla_\theta\langle E\rangle,$$

where θ is the parameters of the model, μ is the learning rate, and $\nabla_\theta\langle E\rangle$ is the derivative of the loss value.

As a better alternative to vanilla gradient decent, we use Tensorflows ADAM optimizer with learning rate $\mu = 0.01$, and otherwise default values, as it is a popular choice in the machine learning community.

D. One-Body Density

To extract a one-body density from a Monte-Carlo simulation, the space is partitioned into a number of bins in an appropriate range where the wave function is large. The bin size can be chosen small to get finer details of the density, but will require more data to mitigate statistical error.

For each particle configuration produced at every Metropolis step, the number of particles coinciding with each bin is checked. The one-body density is then produced by averaging over all Metropolis steps.

IV. RESULTS AND DISCUSSION

If not otherwise specified, the results have been derived using the following parameters:

- Harmonic oscillator frequency $\omega = 1$
- Coulomb interaction strength and shielding constant $\alpha = 1$, $\beta = 0$
- Metropolis step length 1
- Number of Metropolis steps $\mathsf{steps} = 20$
- Batch size of 500
- Training duration of 500 epochs
- Two layer DNN architecture with 32 nodes each

Note that because of Tensorflows multithreading, the training procedure is not deterministic, even when using seed. Because of this, small differences in results will occur from run to run.

A. Pure DNN Model for One Particle In Harmonic Oscillator

1. One Particle in 1D Harmonic Oscillator, ReLU vs tanh

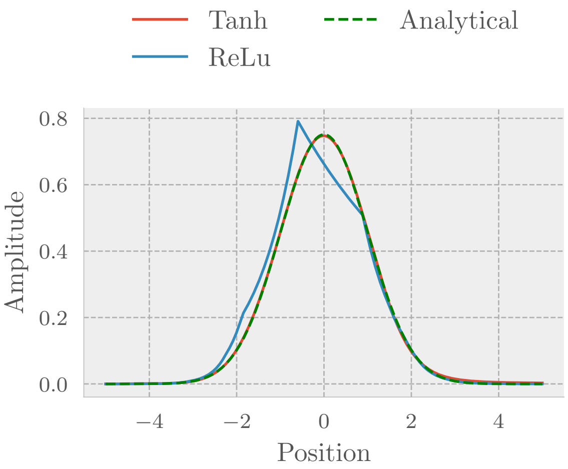

Figure IV.1: Wave function of particle in 1D harmonic oscillator, approximated using DNN with tanh and ReLU activations, respectively. The Metropolis step length was set to 2. The results are plotted against the analytical result

To examine the performance and behavior of neural network based trial wave functions, we start with a pure DNN trial wave function on a simple system of one particle in a 1D harmonic oscillator. Both tanh and ReLU activation functions were used to compare their performance. The Metropolis step length was set to 2 to yield a 50% acceptance rate. The resulting wave function is shown in fig. IV.1. Upon first inspection, the trial wave function with ReLU activation fails spectacularly in approximating the analytical result, while tanh produces a very close-lying approximation.

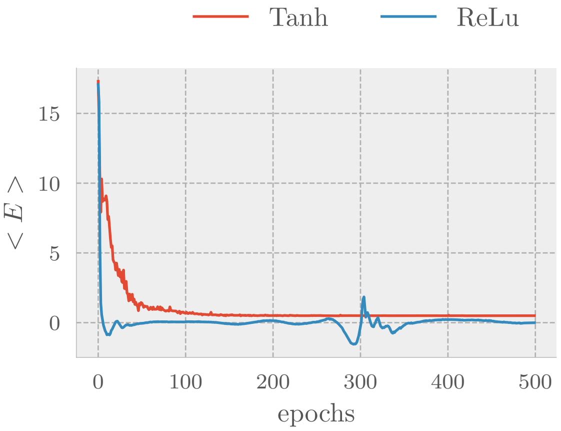

Figure IV.2: Energy estimated from each batch during training of the tanh and *ReLU* model on one particle in 1D harmonic oscillator

The minimization of $\langle E \rangle$ during training for ReLU and tanh can be seen in Figure IV.2, and reveals even more serious problems with ReLU. While the energy of the tanh model smoothly decreases towards the analytical value of $E = 0.5$, the energy of the ReLU model varies wildly, even undercutting the analytical value. This is disastrous, as it violates the variational principle.

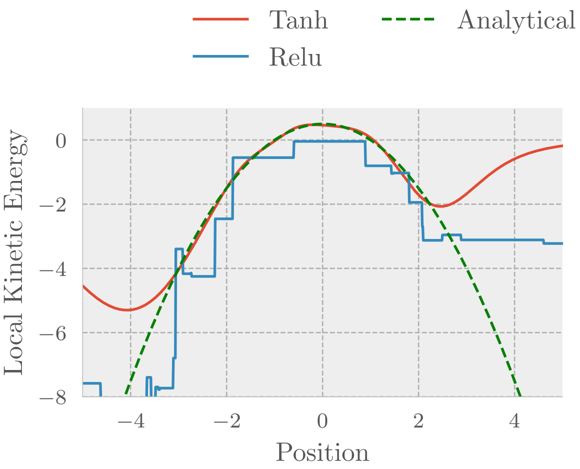

Figure IV.3: Local kinetic energy of the tanh and ReLU model trained on one particle in $1D$ harmonic oscillator

Figure IV.3 shows the kinetic term of the local energy as a function of position, plotted plotted against the analytical result. The local kinetic energy of ReLU model is ill-behaved, characterized by many discrete steps instead of smooth curve. This is likely the cause of its problematic behavior. Since the local kinetic energy relies on the Laplacian of the trial wave function, the use of activation functions with discontinuous derivatives, such as ReLU, appears to produce models which cannot approximate wave functions with appropriate curvature.

Further, Figure IV.3 shows an interesting feature of the tanh model. While it closely approximates the correct local kinetic energy in the center part, it fails for positions far from origo. A possible explanation is that since samples are using the Metropolis algorithm, configurations yielding a small value of the wave function will be sampled less often. As a result, the model may struggle to learn the correct approximation of the kinetic term for these areas. However, this is not a problem for numerical accuracy, since the same configurations that the model struggle to learn will seldom be sampled, and hence their contribution to the expectation values will be negligible.

As the ReLU fails to produce physical results, it is not used in the rest of this paper.

2. *One Particle in 2D and 3D Harmonic Oscillator*

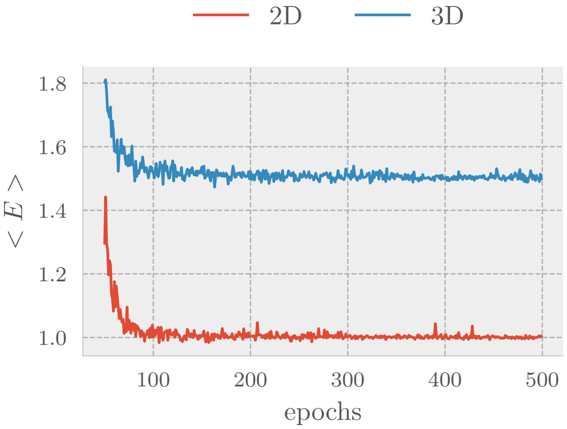

Figure IV.4: Energy estimated from each batch during training of the DNN model on one particle in harmonic oscillator, in $2D$ and $3D$, respectively

The minimization of $\langle E \rangle$ during training of the DNN model of one particle in 2D and 3D harmonic oscillator can be seen in Figure IV.2. Both energies approach the correct ground state energy, $E = 1$ and $E = 1.5$ in two and three dimensions, respectively. The fluctuations in the energies decrease as the training progresses, indicating that the trial wave function approaches the correct ground state, as $\sigma^2 \to 0$ when $\psi_{\text{Trial}} \to \psi_{\text{GS}}$.

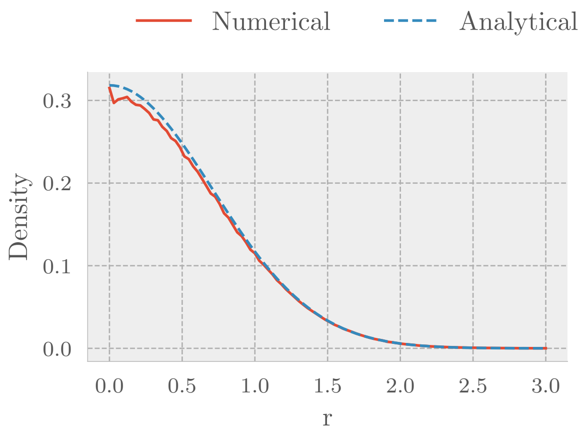

Figure IV.5: Radial one-body density for one particle in 2D harmonic oscillator. The density was calculated using $N = 1 \times 10^6$ samples from the trained DNN model, using 100 bins on the interval $[0, 3]$. It is compared to the analytical result

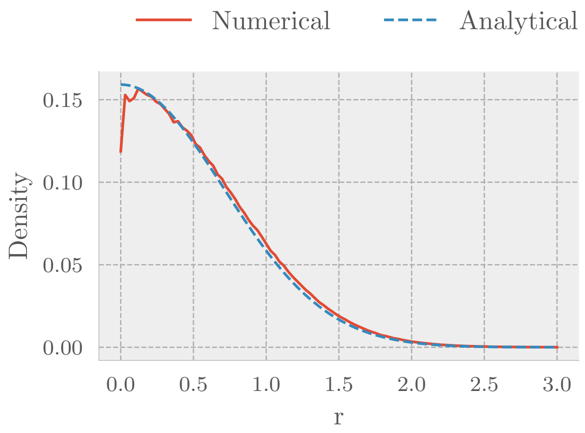

Figure IV.6: Radial one-body density for one particle in 3D harmonic oscillator. The density was calculated using $N = 1 \times 10^6$ samples from the trained DNN model, using 100 bins on the interval $[0, 3]$. It is compared to the analytical result

After the previous training, the radial one-body density is calculated using $N = 1 \times 10^6$ samples generated by the model. The densities are presented in Figure IV.5 and Figure IV.6. Although the approximation is not as close as seen in Figure IV.1, it is still fairly good.

3. Ground State Energies

In Table IV.1, a summary of the estimated ground stated energies of the DNN model trained on the previously discussed systems is presented. Without too much concern for choice of parameters,all models but the ReLU model produces results to accurate to within $1\% - 3\%$ of the analytical values.

	Numerical	Analytical
1D, tanh	0.5015(2)	0.5
1D, ReLU	0.0032(2)	0.5
2D, tanh	1.0015(1)	1.0
3D, tanh	1.5046(1)	1.5

Table IV.1: Summary of the estimated ground state energies of the DNN model trained on the previously discussed systems. The energy was estimated using $N = 1 \times 10^6$ samples

B. RNN-DNN Model for Non-Interacting Bosonic Quantum Dots

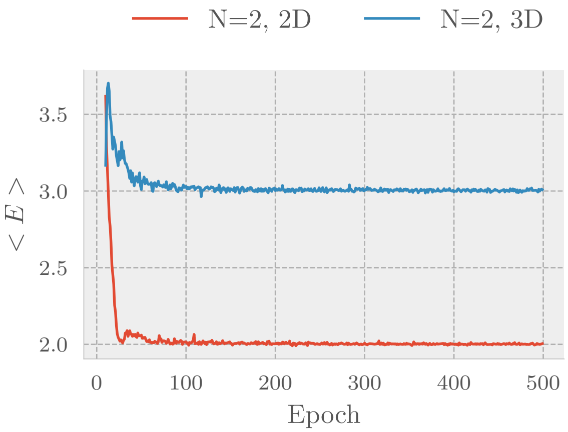

Figure IV.7: Estimate of energy during training of RNN-DNN models. The models were trained on two non-interacting bosons in 2D and 3D harmonic oscillator. The models were trained for 1000 epochs, using a batch size of 500.

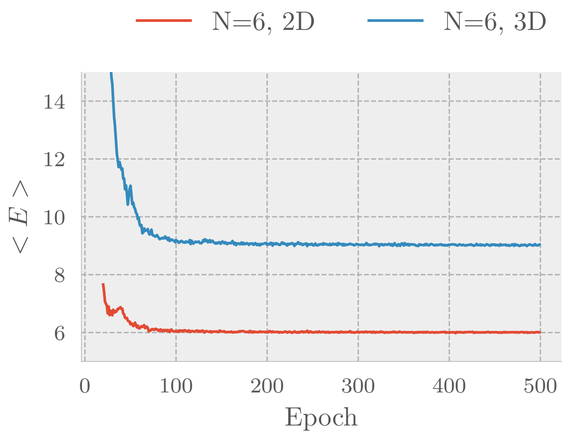

Figure IV.8: Estimate of energy during training of RNN-DNN hybrid models. The models were trained on six non-interacting bosons in 2D and 3D harmonic oscillator. The models were trained for 1000 epochs, using a batch size of 500.

In Figure IV.7 and Figure IV.8, we see the minimization of energy as the RNN-DNN model is trained on various numbers of non-interacting bosons in 2D and 3D harmonic oscillators. In all cases, the model used a hidden state of 5 units, together with a two layer network with 32 nodes each. Although the RNN-DNN uses the hidden state to encode the correlation with previously sampled particles, it has no problem learning the ground state of uncorrelated systems, as seen from the figures. Note that in Figure IV.8, a sudden dip in the energy in the beginning training can be seen for both the models. This dip violates the variational principle. As nothing was done to regularize the trial wave function, it is perhaps unnormalizable during early stages of training, since the initial parameters are random. This will likely cause the metropolis algorithm to fail, as it is not possible to sample from such a distribution. Nevertheless, the model approaches the correct ground state energy eventually.

	Numerical	Analytical
N=2, 2D	2.0017(5)	2
N=6, 2D	6.0044(8)	6
N=2, 3D	3.0021(5)	3
N=6, 3D	9.012(1)	9

Table IV.2: Summary of the estimated ground state energies of the RNN-DNN model in the non-interacting case. The energy was estimated using $N = 1 \times 10^5$ samples

From Table IV.2, we see that the RNN-DNN model attains good accuracy for the non-interacting case, both in 2D and 3D, averaging a relative error of 0.1%.

C. RNN-DNN Model for Two Interacting Bosonic in 1D Quantum Dots

1. *Training*

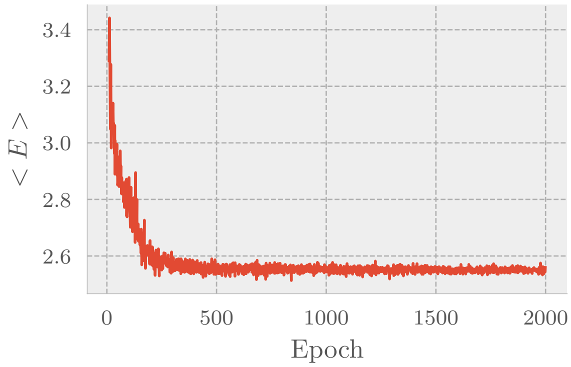

Figure IV.9: Estimate of energy during training of RNN-DNN hybrid model. The model was trained on two interacting bosons in 1D harmonic oscillator. For the Coloumb interaction, a strength value of $\alpha = 1$ and a shielding value of $\beta = 0.1$ was used. The models were trained for 2000 epochs, using a batch size of 2000.

In Figure IV.9, we see the training process of two interacting bosons in 1D harmonic oscillator. The minimization of the energy is a bit more noisy than the previous non-interacting cases, hence the batch size was increased. The model used a hidden state of 5 units, together with a two layer network with 64 and 32 nodes, respectively.

Using $N = 1 \times 10^6$ samples, the ground state energy was estimated to be $E = 2.552(1)$. Compared to $E = 2.5482$ CISD energy using 40 orbitals(courtesy Øyvind Schøyen), the relative error is about 0.1%.

2. One-Body Density

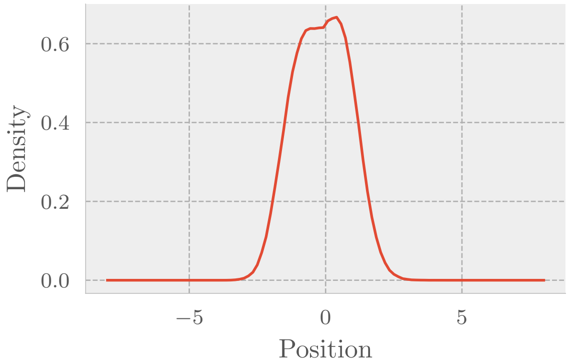

Figure IV.10: One-body density for two interacting bosons in 1D harmonic oscillator. The density was calculated using $N = 4 \times 10^5$ samples and 100 bins on the interval $[-8, 8]$.

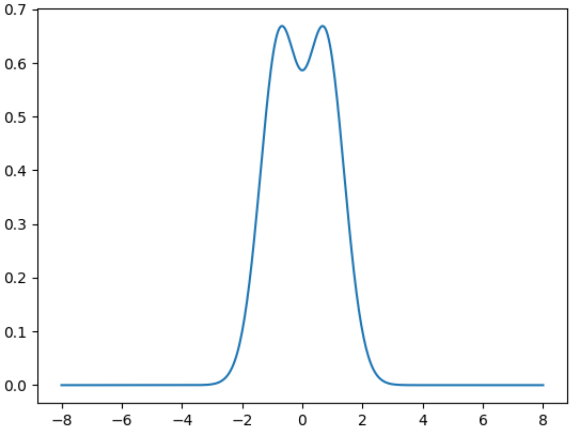

Figure IV.11: One-body density for two interacting bosons in 1D harmonic oscillator using CISD,courtesy Øyvind Schøyen

Figure IV.10 shows the corresponding one body density, which shows the characteristic separation of two peaks caused by the repulsive interaction. Comparing with the one-body density produced with the CISD simulation, the peaks are not as sharply defined. In terms of ground state energy, the RNN-DNN model successfully accounts for the correlation of the system, showing that the hidden state is able to encode the correlation in a meaningful way. However, like other approches in machine learning VMC, such as RBM, it struggles to reproduce the correct one-body density.

3. Conditional Probabilities

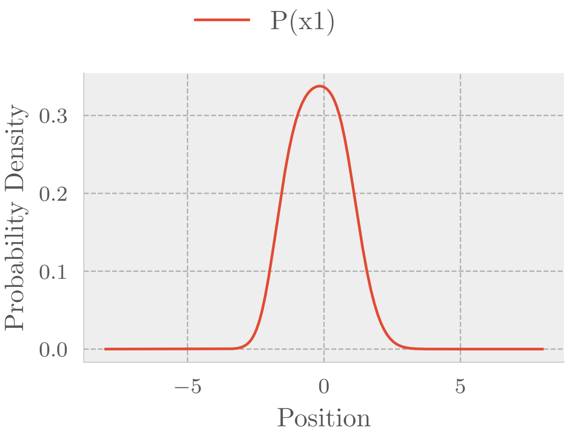

Figure IV.12: Probability density $P(x_1)$ of the first sampled boson for the system with two interacting bosons in 1D harmonic oscillator.

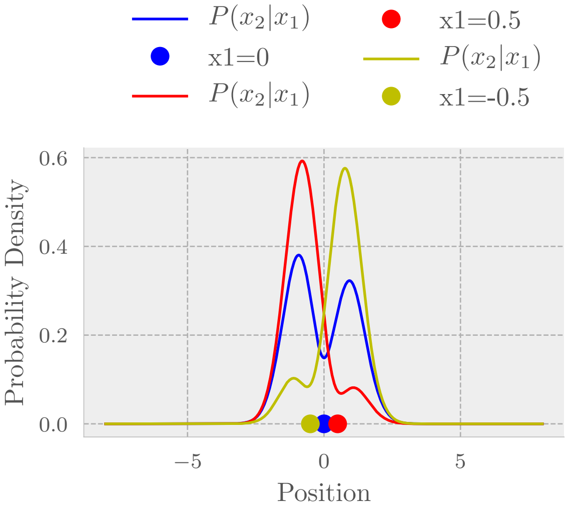

Figure IV.13: Conditional probability density $P(x_2|x_1)$ of the second sampeled boson for the system with two interacting bosons in 1D harmonic oscillator.

In Figure IV.12 and Figure IV.13, we see the conditional probabilities produced by the RNN-DNN model trained on two interacting bosons in 1D harmonic oscillator. In Figure IV.12, we see that the probability density for particle 1 is skewed towards the left. As there is no physical reason for why the first sampled particle is found more often to the left, one should think of the conditional probabilities more as rules for sampling the particles, rather than physical objects.

In fact, given $P(x_1, x_2)$, there is no unique way of factoring it to conditional probabilities:

$$\begin{aligned} P(x_1, x_2) &= P(x_1)P(x_2|x_1) \\ &= f(x_1)P(x_1)\frac{P(x_2|x_1)}{f(x_1)} \\ &= \tilde{P}(x_1)\tilde{P}(x_2|x_1), \end{aligned}$$

where $f(x_1)$ is an arbitrary function.
In Figure IV.13, we see however that some physical meaning is retained. The probability density of x_2 conditioned on x_1 acts as to avoid the probability density of x_1. Given x_1, the probability of sampling x_2 close to x_1 is relatively small compared to sampling it further away. This is a reflection of the repulsive potential, which works to push the particles apart.

4. Sampling from Conditional Probabilities

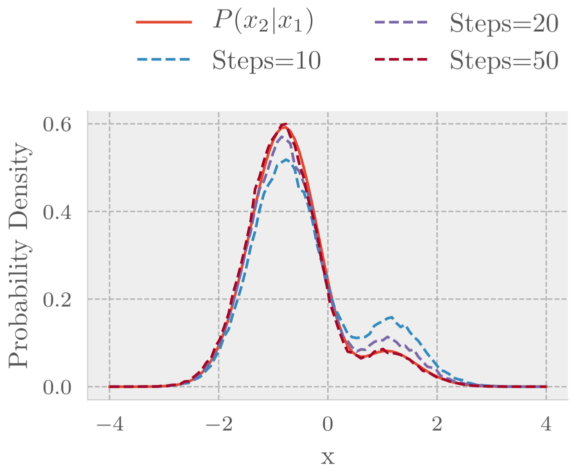

Figure IV.14: Probability density produced with Bruteforce Metropolis, using a step length of 1 and various number of steps. The target probability distribution is the conditional distribution $P(x_2|x_1 = 0.5)$, as discussed in the previous section.

Next, we examine if Brute force Metropolis sampling manages to sufficiently sample the distributions. In Figure IV.14, the target probability density is $P(x_2|x_1 = 0.5)$ produced by the RNN-DNN model trained on the system as previously discussed. The densities were estimated using Bruteforce Metropolis with a step length of 1 and varying number of thermalization steps. For a low number of steps, such as 10, it can be seen from the figure that the estimated density does not match the one produced by the model. Since the random walkers start off uniformly, they need a sufficient amount of steps so that they don't end up getting stuck in low probability areas, as seen to the right in the figure. If this happens, we fail to sample variables that are distributed as our model dictates. This is remedied by taking ample amount of steps.

D. RNN-DNN Model for Two Interacting Bosons In Higher Dimensions

1. Ground State Energy

	RNN-DNN	DMC[7]
2D	3.014(3)	3.000 00(1)
3D	3.751(1)	3.730 123(3)

Table IV.3: Estimated ground state energies of the RNN-DNN model train on two interacting bosons in 2D and 3D harmonic oscillator, respectively. In both cases, 10 hidden units and two layers of 64 and 32 nodes were used. The batch size was 4000 and the models were trained for 1000 epochs. The energy was estimated using $N = 1 \times 10^5$ samples.

From Table IV.3, we see the estimated ground state energy of the RNN-DNN model trained on two interacting bosons in 2D and 3D harmonic oscillator. Taking the DMC calculation as ground truth, the numerical error is under 1% in 2D and 3D, as with the 1D case.

2. One-Body Density

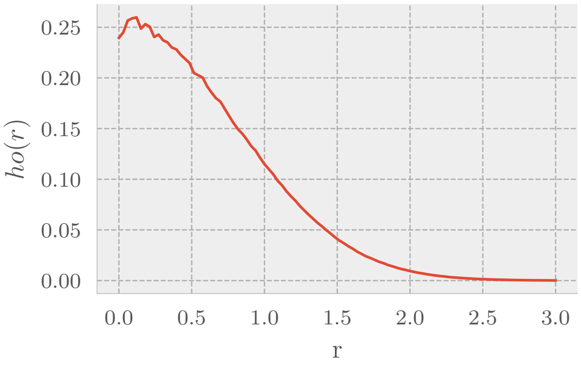

Figure IV.15: Radial One-body density for two interacting bosons in 3D harmonic oscillator. The density was calculated using $N = 1 \times 10^6$ samples and 100 bins on the interval $[0, 3]$.

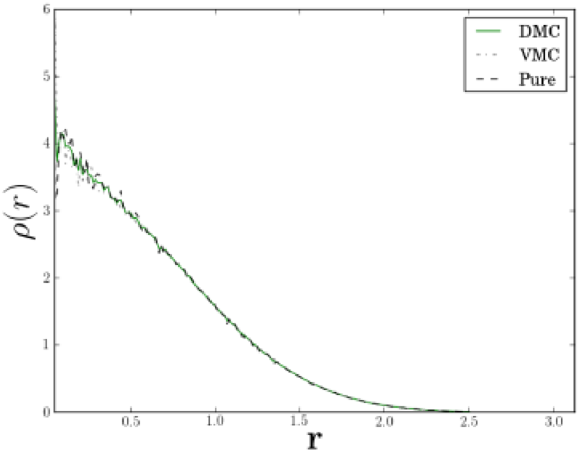

Figure IV.16: Radial One-body density for two interacting bosons in 3D harmonic oscillator. [7]

In Figure IV.15 and Figure IV.16 we see the radial one-body density of two interacting bosons in 3D harmonic oscillator calculated using our RNN-DNN model and DMC, respectably. Note that technically, the DMC calculation was performed on a two electron system, but since this state is symmetrical with respect to spatial coordinates, it is comparable to our bosonic system. In addition, the density is not scaled.

Since the figures are produced independently, they are hard to compare. However, some general features can be seen: 1/3 of the max amplitude occur around $r = 1$, and the density have almost completely diminished around $r = 2$. On the other hand, the density produced by the RNN-DNN model is not as linear towards the origin as the DMC density, again indicating the our model may struggle reproducing the correct one-body density.

V. CONCLUSION

When using neural network based trial wave function to approximate the solution of Hamiltonians defined on continuous space, it is necessary to use activation function with continuous derivatives, such as tanh, rather than Relu with discontinuous derivatives. As seen from Figure IV.3, using ReLu activation resulted in a local kinetic energy that did not match the analytical, but rather a piece-wise constant function.
Moving to the RNN-DNN model, the model perform well in the non-interacting case, both in 2D and 3D and for two and six particles. In all cases, the relative numerical error was of order 0.1%, as seen in Table IV.2. Generally, VMC often achieves results accurate to machine precision in the case of non-interacting systems. This is because the trial wave function is carefully picked to include the analytical ground state in its function space. In the case of the RNN-DNN model, the trial wave function is much more flexible and completely general. It essentially learns the solution from scratch, without having any prior information about the system other than the Hamiltonian.
When targeting two interacting bosons in 1D, 2D and 3D, it was required to increase the complexity of the model, up the batch size and extend the training in order to keep the error to a minimum. This was likely due to the increased complex because of the correlation introduced by the interaction. Although the result was not as accurate as in the non-interacting case, the relative error was still under 1%, as seen in Table IV.3. While the model is able to obtain good accuracy for the ground state energy, it seem to struggle a bit more approximating the correct one-body density. This can be seen by comparing Figure IV.10 and Figure IV.11.In the 1D case, the one-body density produced by the RNN-DNN model was somewhat asymmetrical, and the separation of the peaks were not as sharply defined as the benchmark. It is a common problem with machine learning VMC approaches, such as RBM, that the one-body density does not match the physical. This is perhaps the case for our model as well.

VI. FUTURE WORK

This paper focused on testing whether or not it was possible to create a model with auto-regressive properties for bosonic quantum dots. Having achieved promising results, the way forward is to test the method for larger systems of interacting particles. This was not done in this paper because of a lack of appropriate benchmarks to compare with. Testing on bigger systems, it will be important determine how the number of parameters should be increased in order to keep results accurate. In doing this, it will also be revealed how the computational complexity of the model scales with increasing system size, and if having the auto-regressive property

leads to any significant speedup with respect to more classical VMC approaches.
In testing bigger systems, the encoding of the correlation done by the RNN will likely be a more challenging task. Other RNN architectures, like GNU and LSTM, should be explored. Other optimization techniques, other than default ADAM, should also be explored. Optimization methods involving approximation of the curvature of the loss value has been shown to be effective [2]. Also other methods for sampling, such as importance sampling, should be implemented for more efficiency.

[1] H. Flyvbjerg and H. G. Petersen, The Journal of Chemical Physics **91**, 461 (1989), https://doi.org/10.1063/1.457480.

[2] D. Pfau, J. S. Spencer, A. G. de G. Matthews, and W. M. C. Foulkes, “Ab-initio solution of the many-electron schrödinger equation with deep neural networks,” (2019), arXiv:1909.02487 [physics.chem-ph].

[3] M. Hibat-Allah, M. Ganahl, L. E. Hayward, R. G. Melko, and J. Carrasquilla, “Recurrent neural network wavefunctions,” (2020), arXiv:2002.02973 [cond-mat.dis-nn].

[4] O. Sharir, Y. Levine, N. Wies, G. Carleo, and A. Shashua, Physical Review Letters **124** (2020), 10.1103/physrevlett.124.020503.

[5] P. Mehta, M. Bukov, C.-H. Wang, A. G. R. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, ArXiv e-prints (2018), arXiv:1803.08823 [physics.comp-ph].

[6] J. X. Zhi-Qin, “A note of using tensorflow to code laplacian operator in high dimenion,” (2020).

[7] B. Samseth, *Learning Correlations in Quantum Mechanics with Neural Networks*, Master’s thesis (2019).