<u>**ISTANBUL TECHNICAL UNIVERSITY**</u>

<u>**FACULTY OF SCIENCE AND LETTERS**</u>

Graduation Project

Machine Learning and Non-linear Schrödinger Equation

Hüseyin Talha Şenyaşa

Department	**: Physics Engineering**
Student ID	**: 090120132**
Advisor	**: Assoc. Prof. A. Levent Subaşı**

FALL 2017

Summary

We train an artificial neural network to estimate the ground state energy of a one-dimensional Bose-Einstein condensate in different type of potentials including random. Such a system can be described by the solution of a non-linear Schrödinger equation also called a Gross-Pitaevskii equation. We also use the method for the inverse problem of predicting the non-linearity parameter using the ground state density profile for a given harmonic trapping potential.

Contents

1 Introduction and Motivation

In quantum mechanical systems one must obtain the wave function to determine physical properties of the system. For single particle systems, obtaining the wave function is easy compared to many body systems. The solution can be even harder if the described system involves interaction between particles because such differential equations may have nonlinear terms. In nonlinear case, there are no general rule to solve and they are treated individually in most cases. If there are no analytic solution exists and approximation is not on the table then the only chance is numerical solution of the equation to obtain the wave function, and then one can determine the physical properties of the system. However, machine learning applications in recent years showed that physical properties of the system can be predicted without solving these equations.

The reason why machine learning has such a capability is that artificial neural networks used in machine learning can approximate any continuous function within desired accuracy. This means that if we take $\epsilon > 0$ as desired accuracy, $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ (bold means vector) output of the network and $\boldsymbol{g}(\boldsymbol{x})$ real value of the function, then it is guaranteed that there exists a network that satisfies the relation $||\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})|| < \epsilon$. From this point of view, a process or calculation that can be thought of as a function can be represented by a corresponding neural network that mimics this function in desired accuracy [1]. With this in mind, neural networks have the potential to learn general functions and be exploited for their advantages. Approximate value of the quantity can be determined for different scenarios.

Many different kind of applications of machine learning have already been implemented in physics[1]. For example, In [3, 4] machine learning is applied to quantum many body problems. A machine learning method called Unsupervised Learning to detect patterns in big datasets is used for discovering phase transitions [5]. There are also developed techniques in machine learning inspired from physics such as quantum tensor networks [6]. Relation between physics and machine learning also caused foundation of a new branch called Quantum Machine Learning which aims to implement a quantum software to make machine learning faster than its classical version [7].

In [8], machine learning approach is applied to a 2D Schrödinger equation. The authors built a convolutional deep neural network, and trained it to predict

[1]For a detailed list, see [2]

ground state energy of the system under four different confining potentials including random potential. Their study showed that machine learning is a promising alternative in electronic structure calculations of quantum systems. In our study inspired from the article mentioned above, we apply machine learning method to the nonlinear Schrödinger equation also known as Gross-Pitaevskii equation to predict ground state energy of a Bose-Einstein condensate with random interaction parameter at absolute zero temperature under six different one dimensional trapping potentials including random.

2 Gross Pitaevskii Equation

A Bose-Einstein Condensate (BEC) at zero temperature is described by Gross Pitaevskii equation (GPE) also known as non-linear Schrodinger Equation (NLSE). It is mean field approximation of a quantum many body system which the hamiltonian of the system is given by;

$$\hat{H} = \sum_{i=1}^{N} \left(\frac{\boldsymbol{p}_i^2}{2m} + V(\boldsymbol{r}_i) \right) + \frac{1}{2} \sum_{i=1}^{N} \sum_{j \neq i}^{N} U(|\boldsymbol{r}_i - \boldsymbol{r}_j|) \tag{1}$$

where $\boldsymbol{p}_i$ i^{th} atom's momentum, $\boldsymbol{r}$ is position vector, m is mass, V is external potential and U is interaction between i^{th} and j^{th} atoms. Since the ground state energy is the only possible minimum energy, one can minimize this hamiltonian in order to obtain ground state energy. To do that, the mean field approximation is utilized to represent all bosons with the same wave function since the condensate is at zero temperature which allows to assume that all bosons are at the ground state [9]. The constraint in the minimization step involves satisfying the normalization condition that is given as,

$$\int |\psi(\boldsymbol{r})|^2 \, \mathrm{d}^3\boldsymbol{r} = N \tag{2}$$

where N is the number of particles in the system. This condition equals to minimizing the thermodynamic potential, free energy $F = E - \mu N$ such that $\delta F = 0 = \delta E - \mu \delta N = 0$ where μ is chemical potential [9]. If we plug in the corresponding equations to this expression it becomes,

$$F(\psi) = \int \psi^* \hat{H} \psi \, \mathrm{d}^3\boldsymbol{r} - \mu \int |\psi|^2 \, \mathrm{d}^3\boldsymbol{r} \tag{3}$$

Therefore, the problem can be stated as minimization of the above equation [10]. Applying variation method to this equation while treating ψ^* and ψ as independent objects and using $U(|\boldsymbol{r}_i - \boldsymbol{r}_j|) = g\delta(\boldsymbol{r}_i - \boldsymbol{r}_j)$ one can obtain Gross

Pitaevskii Equation in stationary form as,

$$\frac{-\hbar^2}{2m}\nabla^2\psi + V(\boldsymbol{r})\psi + g|\psi|^2\psi = \mu\psi \tag{4}$$

where $\hbar$ is Planck constant, $\psi(\boldsymbol{r})$ is the wave function, ∇^2 is the Laplacian operator, g is interaction parameter and it is defined as

$$g = \frac{4\pi\hbar^2 a_s}{m} \tag{5}$$

where, a_s is the s wave scattering length. Nonlinearity of the equation is caused by the cubic term $g|\psi|^2$ which represents the interactions between bosons. The $|\psi|^2$ term is interpreted as density, thus; there occurs an energy contribution caused by the mean field interactions. If there is no interaction GPE reduces to the Schrödinger Equation (SE) and becomes a linear equation. By definition, g can be positive or negative. If $g > 0$, it represents repulsive interaction, and if $g < 0$, it represents attractive interactions [11].

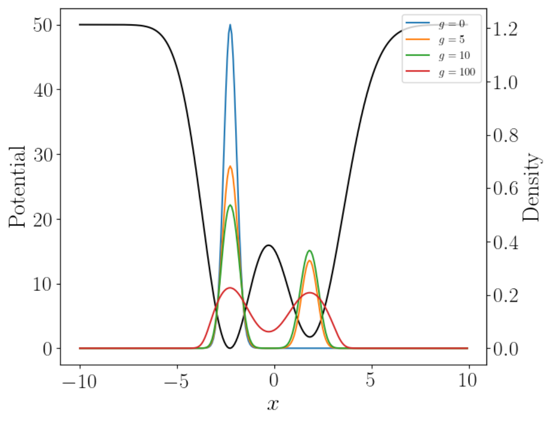


Figure 1: Density under one dimensional double inverted gaussian potential with different interaction parameters. Here density becomes more flatten as the interaction parameter increases.

The time evolution of the system must occur according to the Schrödinger Equation, therefore; the time dependent GPE can be written as,

$$i\hbar\frac{\partial\Psi(\boldsymbol{r},t)}{\partial t} = \hat{H}\Psi(\boldsymbol{r},t) \tag{6}$$

where t is time and $\Psi(\boldsymbol{r},t) = \psi(\boldsymbol{r})e^{-i\mu t/\hbar}$. If it is rewritten in open form,

$$i\hbar\frac{\partial\Psi(\boldsymbol{r},t)}{\partial t} = \left[\frac{-\hbar^2}{2m}\nabla^2 + V(\boldsymbol{r}) + g|\Psi(\boldsymbol{r},t)|^2\right]\Psi(\boldsymbol{r},t) \tag{7}$$

The ground state energy of the system is given in Eq. (3) which is,

$$\langle E\rangle = \int \Psi^* \hat{H} \Psi \, \mathrm{d}^3\boldsymbol{r} \tag{8}$$

$$E = \int \left(\frac{\hbar^2}{2m} |\nabla\Psi|^2 + V|\Psi|^2 + \frac{g}{2}|\Psi|^4 \right) \mathrm{d}^3\boldsymbol{r} \tag{9}$$

Here, the terms represent kinetic, potential and interaction energy respectively.[2] Direct relation between chemical potential and energy can be obtain from integration of the Eq (7),

$$\mu = \frac{E_{kin} + E_{pot} + 2E_{int}}{N} \tag{10}$$

In the non-interacting case this expression leads to energy per particle and if the potential does not depend on time the total energy of the system is conversed.

2.1 Reduction of Dimension

Since we are going to work in one dimension, we must express 1D GPE by reducing the dimensionality of the 3D GPE. This reduction of dimension step is done under anisotropic harmonic trapping potential given by;

$$V(x, y, z) = \frac{1}{2} m(\omega_x^2 x^2 + \omega_y^2 y^2 + \omega_z^2 z^2) \tag{11}$$

where ω_x,ω_y,ω_z are angular frequencies. The confinement of the physical dynamics of the system can be performed by making angular frequencies asymmetric such that $\omega_z, \omega_y \gg \omega_x$ and keeping their energy order much greater than condensate's energy $\hbar(\omega_z\omega_y)^{1/2} \gg \mu$. Now the dynamics of the system only develop in one dimension and it can be described with a corresponding one dimensional GPE equation [12]. To obtain one dimensional GPE one can rewrite wave function,

$$\psi(x, y, z, t) = \psi_x(x, t)\psi_y(y)\psi_z(z) \tag{12}$$

ψ_y and ψ_z are the ground state solution of the transverse potentials;

$$\begin{aligned} \psi_y(y) &= \frac{1}{(\pi l_y^2)^{1/4}} e^{-y^2/2l_y^2} \\ \psi_z(z) &= \frac{1}{(\pi l_z^2)^{1/4}} e^{-z^2/2l_z^2} \end{aligned} \tag{13}$$

where $l_y = \sqrt{\hbar/m\omega_y}$ and $l_z = \sqrt{\hbar/m\omega_z}$ are the harmonic oscillator lengths that

[2]Contribution from the interaction energy is divided by two to eliminate double counting while pairing bosons.

related to the density distribution of the condensate. Normalization of the wave function is chosen such that $\int \psi_y(y)\,\mathrm{d}y = \int \psi_z(z)\,\mathrm{d}z = 1$, and $\int \psi_x(x,t) = N$. If we plug in the factorized wave function to Eq. (7), then, the equation becomes;

$$\mu' \psi_x = \frac{-\hbar^2}{2m}\frac{\mathrm{d}^2\psi_x}{\mathrm{d}x^2} + \frac{1}{2}m\omega_x^2 x^2 \psi_x + g'|\psi_x|^2\psi_x \tag{14}$$

where μ' and g' one dimensional effective chemical potential and interaction strength respectively

$$\mu' = \mu - \frac{\hbar}{2}(\omega_y - \omega_z), \quad g' = \frac{g}{2\pi l_y l_z} \tag{15}$$

Here we showed that how interaction parameter and chemical potential are effected when dimension of the 3D GPE is reduced. We are going to use these effective quantities in our study. We also directly used the harmonic potential for simplicity but Eq (11) can be replaced by another equation such as,

$$V(x,y,z) = V(x) + \frac{1}{2}m(\omega_x^2 x^2 + \omega_y^2 y^2 + \omega_z^2 z^2) \tag{16}$$

where $V(z)$ can be optical potential for instance [13]. Since relation between potential and dimension reduction is not in our scope, we directly change the potential expression with a generic one and accept the one dimensional GPE as,

$$\frac{-\hbar^2}{2m}\frac{\mathrm{d}^2\psi_x}{\mathrm{d}x^2} + V(x)\psi + g|\psi|^2\psi = \mu\psi \tag{17}$$

and from now on, we refer to Eq. (17) as GPE.

2.2 Analytic Solution and Approximation

There is no general solution of GPE. Known analytic solutions exist only for few cases and lack of analytic solution is also one of the main motivation of the application of machine learning technique. In our study, infinite well and harmonic potential with zero interaction parameter has analytic solutions. Here we give the ground state wave solution and energy of the harmonic potential in non-interacting case which is also solution of the SE,

$$\psi(x) = (\frac{m\omega}{\pi\hbar})^{1/4} \exp^{-m\omega x^2/\hbar} \tag{18}$$

$$E = \mu = \frac{1}{2}\hbar\omega \tag{19}$$

respectively. Comparison of the numerical solution and analytic solution in given in the figure.

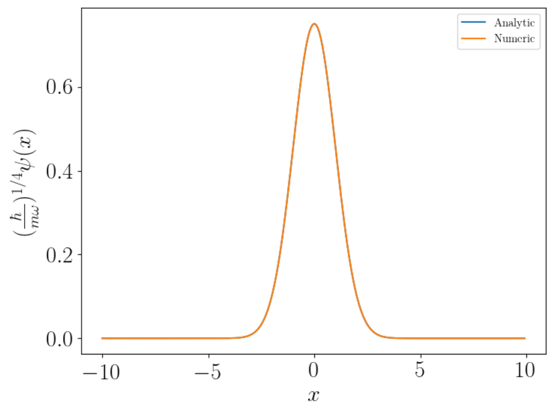


Figure 2: **COMMENT**

GPE generally solved by numerically or by approximation such as variational calculation or Thomas-Fermi approximation. In the following section we briefly introduce Thomas-Fermi approximation and then we add comparison with numerical solutions.

2.2.1 Thomas Fermi Approximation

It is said that the second derivative term with respect to position represents the kinetic energy. When the potential and the interaction energy are dominant compared to the kinetic energy, the kinetic term can be neglected. This situation occurs when the condensate is large adequately and the interaction between bosons is repulsive [9]. In another perspective, neglecting the kinetic term equivalent to making an assumption that there is no difference between the energy requirements to add a particle at arbitrary points [10]. When the kinetic term is dropped, the new equation can be written as;

$$V(x)\psi(x) + g|\psi(x)|^2\psi(x) = \mu\psi(x) \tag{20}$$

this equation is analytically solvable and the solution is given by,

$$n(x) = |\psi(x)|^2 = \begin{cases} (\mu_{TF} - V)/g & \text{if } |x| \le x_{TF} \\ 0 & \textit{otherwise} \end{cases} \tag{21}$$

where x_{TF} is called Thomas-Fermi Length and $n(x)$ is density. The density defined in this range and cannot be negative, therefore; if the normalization condition Eq. (2) is applied then the equation reads,

$$\int_{-\infty}^{\infty} |\psi|^2 \,\mathrm{d}x = \int_{-x_{TF}}^{x_{TF}} \frac{(\mu - V)}{g} \,\mathrm{d}x = N \tag{22}$$

With this equation, density of the system can be obtain via invoking boundary conditions. For an exact analytic expression example, one dimensional harmonic potential can be used,

$$\frac{1}{g}\left[\int_{-x_{TF}}^{x_{TF}} \mu \,\mathrm{d}x - \int_{-x_{TF}}^{x_{TF}} \frac{1}{2} m\omega^2 x^2 \,\mathrm{d}x\right] = N \tag{23}$$

$$\frac{2\mu x_{TF}}{g} - \frac{m\omega^2 x_{TF}^3}{3g} = N \tag{24}$$

From boundary conditions,

$$\mu = V(x_{TF}) = \frac{1}{2} m\omega^2 x_{TF}^2 \tag{25}$$

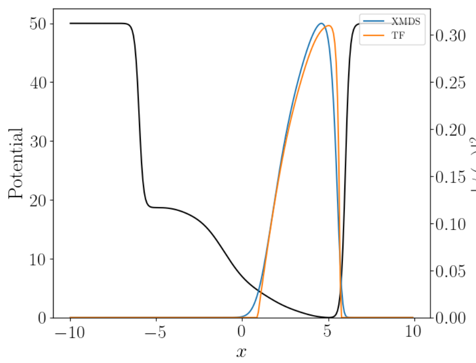
If we combine Eq. (24) and Eq. (25) the equations reads,

$$\frac{m\omega^2 x_{TF}^3}{g} - \frac{m\omega^2 x_{TF}^3}{3g} = N \tag{26}$$

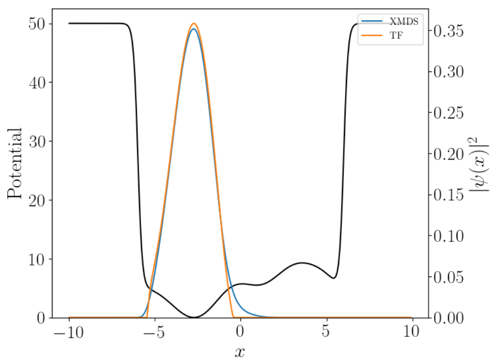
$$\frac{4}{3}\left(\frac{2\mu}{m\omega^2}\right)^{1/2} \frac{\mu}{g} = N \tag{27}$$

$$\mu = \left(\frac{9}{32}(N\omega g)^2 m\right)^{1/3} \tag{28}$$

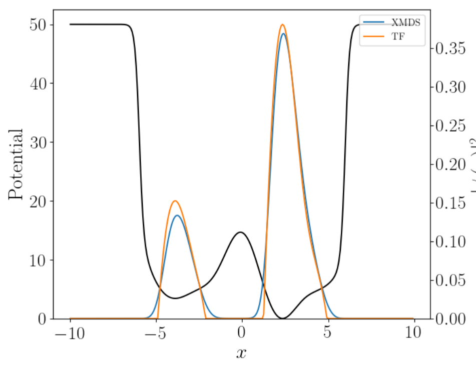
Here we give comparison of densities obtained by numerical solution and Thomas-Fermi approximation for different potential types.



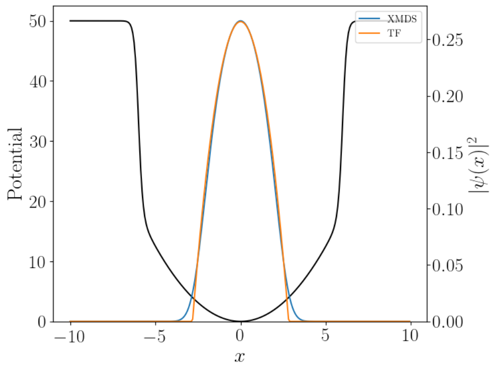
(a) Random Potential Type 1



(b) Random Potential Type 2



(c) Random Potential Type 3



(d) Harmonic Potential

Figure 3: Here the potential types are labeled. The interaction parameter g is 15.

As shown in Fig 3d, the interaction parameter is great enough to use Thomas-Fermi approximation but since the density is determined by the Eq (22) the characteristic must be same with the potential used. This situation can be seen vividly in Fig 3c at near zero region.

2.3 Numerical Solution and Potential Generation Process

The dataset -which will be defined in detail later- generation step is divided into two main parts and implemented independently. The first one is generating desired potential and the second one is numerical solution of the GPE under this potential by giving the generated potential to the numerical solution framework. The detailed description of the potential generation process is given in section 2.3.2.

In numerical solution part, we use a framework called XMDS [14], implemented specifically to solve differential equation systems with well optimized numerical methods. In this framework partial differential equation systems can be described by a markup language called XML. When equation system is declared properly, XMDS produces a source code written in C++ that solves the equation with specified numerical method. Because of modularity in our implementation, the framework only solves the equation by supplied parameters, the framework does not generate anything internally such as potential, even the scaling factors are supplied externally. Such a modularity has an huge advantage such that changing numerical solution framework does not effect potential generation step, therefore; the work to change numerical solution framework is minimized. The only requirement is implementation of input output operations. By using this advantage, we also use another numerical solution framework called GPELab [15] implemented in Matlab to compare solutions' consistency and effect of scaling.

2.3.1 Scaling

The scaling of GPE is generally done according to potential type and there are more than one scaling conventions **REFS**. In this section we use a more general approach to scale GPE since the solutions will be numerical and we briefly would like to investigate how different scalings affect the solutions' consistency. To do that, the dimensionless quantities are introduced with variable coefficients so that scaling coefficient controlled by these variables.

First we define dimensionless potential, and then we make the length dimensionless,

$$\overline{V}(x) \equiv \frac{V(x)}{\gamma E_0}, \qquad \widetilde{x} \equiv \frac{x}{\beta L}$$

Here γ and β positive real numbers. E_0 is in energy unit and L is in length and they are defined respectively as;

$$E_0 = \frac{\hbar^2}{2m}$$

The potential can also be transformed in length scale such that;

$$\widetilde{V}(\widetilde{x}) \equiv \overline{V}(\beta L x)$$

If these transformations are pluged into the Eq. (17) it becomes,

$$\frac{-\hbar^2}{2m\gamma E_0}\frac{1}{\beta^2 L^2}\frac{d^2\psi}{d\widetilde{x}^2} + \widetilde{V}(\widetilde{x})\psi + \frac{g}{\gamma E_0}|\psi|^2\psi = \frac{\mu}{\gamma E_0}\psi \tag{29}$$

To obtain final form, we define dimensionless chemical potential, wave function and interaction parameter respectively.

$$\widetilde{\mu} \equiv \frac{\mu}{\gamma E_0}, \qquad \widetilde{\psi} \equiv \psi\sqrt{\frac{\beta L}{N}}, \qquad \widetilde{g} \equiv \frac{gN}{\gamma E_0 \beta L}$$

To control scaling coefficients we set the coefficient of the kinetic term to an arbitrary positive real number α

$$\frac{\hbar^2}{2m\gamma E_0}\frac{1}{\beta^2 L^2} = \alpha,$$

Now the scaling of GPE can be controlled by α and β only.

$$-\alpha\frac{d^2\widetilde{\psi}}{d\widetilde{x}^2} + \widetilde{V}(\widetilde{x})\widetilde{\psi} + \widetilde{g}|\widetilde{\psi}|^2\widetilde{\psi} = \widetilde{\mu}\widetilde{\psi} \tag{30}$$

Changing β directly effects the length scale and it requires extra transformations in potential generation steps, therefore; we are going to change only α to see the effect of scaling by comparing results. An example of setting the scale coefficients is shown in appendix A.

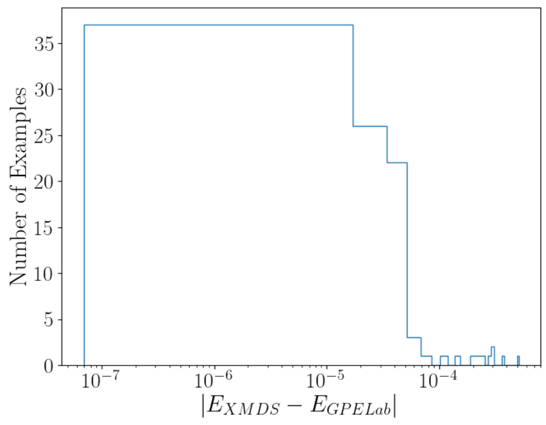


Figure 4: Energy results comparison of different frameworks, XMDS and GPELab

2.3.2 Potential generation

In potential generation, we built a modular structure such that the algorithms to generate potentials are independent from the restrictions forced by numerical techniques such as boundary conditions or scaling. A python class object responsible for handling this process. This class is constructed with suitable parameters such as "number of points", "width" etc. The generated potentials are sent to another method supplied by class which re-scales, and applies an envelope function to ensure that potential goes to numerical limit at boundaries which is given as,

$$\begin{aligned} V(x < x_l) &= V_0 \\ V(x > x_r) &= V_0 \end{aligned} \tag{31}$$

To handle these conditions, we define two envelope functions given by the following expressions,

$$\text{Env}_{LR}(x) = [(1 + \tanh(\beta(x + x_L))) + (1 - \tanh(\beta(x + x_R)))]/2 \tag{32}$$

$$\text{Env}_M(x) = 1 - \text{Env}_{LR}(x) \tag{33}$$

where x_L and x_R are bounds given in the Table (1) and the plot is given in Fig. 5.

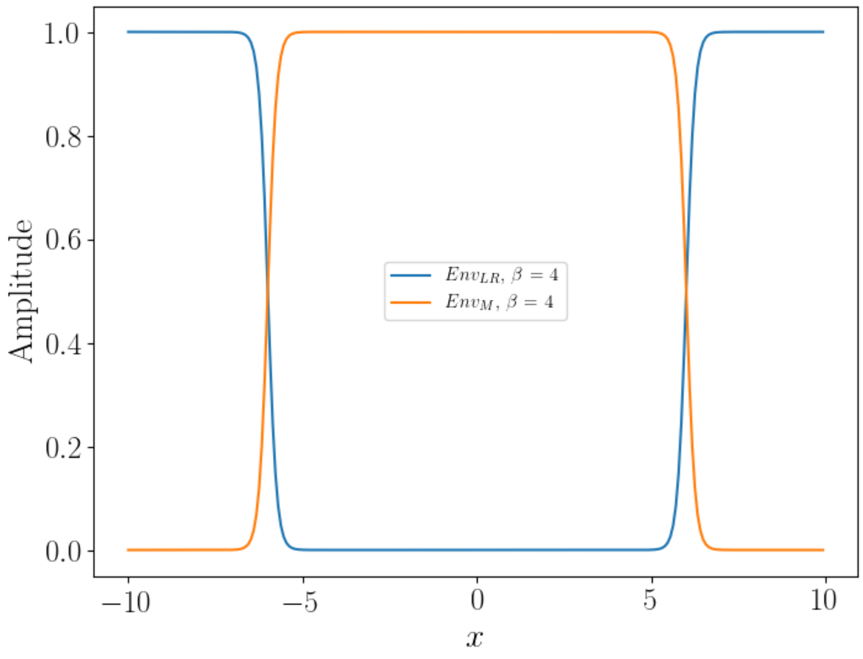


Figure 5: Envelope Function

As shown in Fig. 5 Env_M neutralizes the values while x goes to boundaries and only the region in the middle survives. For Env_{LR} the scenario is vice versa. Simultaneous application of these functions plus re-scaling brings the potential desired form shown in Fig. 6

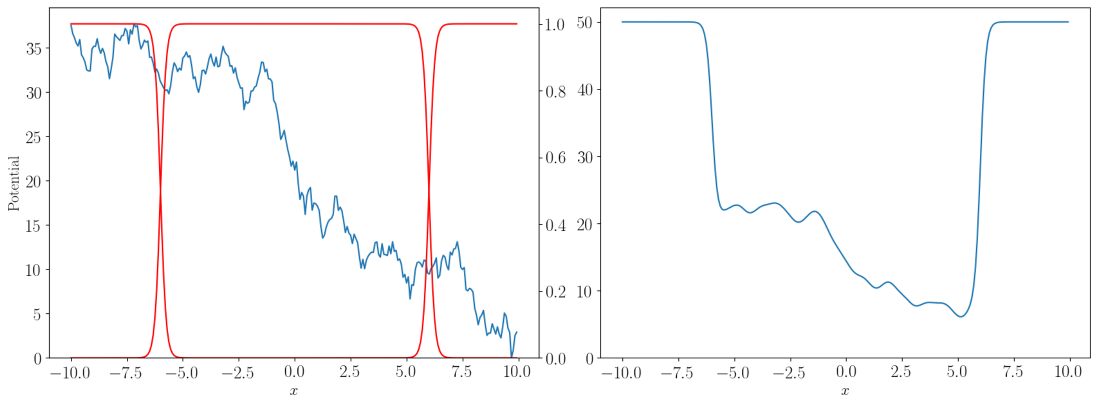


Figure 6: Envelope functions are applied to a random potential. After that, a gaussian filter is applied for smoothness.

2.3.3 Potential Types

Bose-Einstein condensates are generally worked with harmonic trapping potential, but we use six different types of potential in our study. The first three potentials are infinite well, harmonic, and double inverted gaussian potential. Their parametric details are given in Table 1. and the other three are random potentials generated by different algorithms. We investigate training results first individually so that type of the potential in both training and prediction process are same, then we do a cross scenario in that case the type of the potential in training and prediction are different.

2.3.4 Random Potential Generation

To be able to observe the effect of random potential generation to the results we use three different random potential generation algorithms. The first one is random walk with random step size such that the first value of the potential array is initialized with a random number. After that, another random number is added to this value to obtain the next element of the array and so on. The distribution of the random numbers is gaussian in this process. The resultant array is not guaranteed to be smooth. Gaussian filter is applied with a random sigma value to the potential array to make it smooth.

Algorithm 1 RandomPotential1

```
1: procedure RandomPotential1
2:     $Points = GaussianDistributedRandomPoints()$
3:     $Len = Length(Points)$
4:     $Potential[0] = Points[0]$
5:     for i = 0 to $Len - 1$ do
6:         $Potential[i+1] = Potential[i] + Points[i]$
7:     $Potential = GaussianFilter(Potential, \sigma)$
```

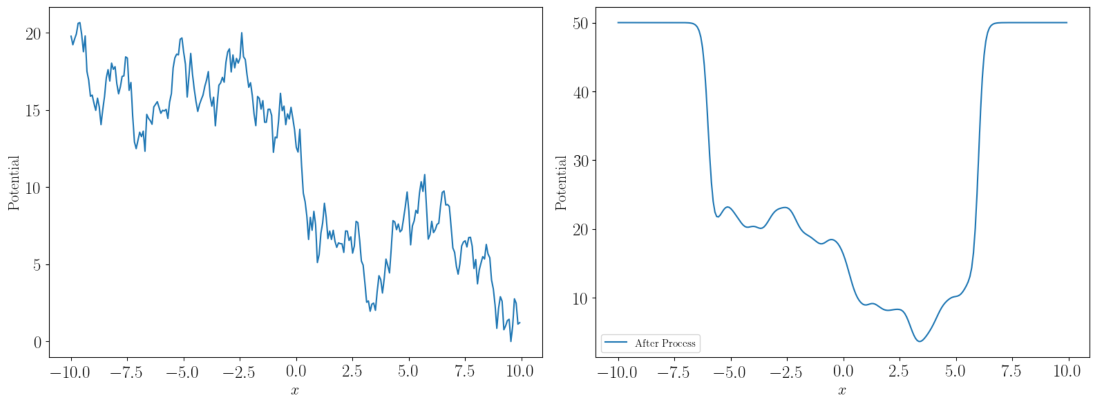


Figure 7: Random Potential 1 Before and After Process

Table 1: Potentials

Potential	Analytic Form / Explanation	Parameters	Description	Min	Max
Envelope	$\mathrm{Env}_{LR}(x) = [(1+\tanh(\beta(x+x_L))) + (1-\tanh(\beta(x+x_R)))]/2$	x_L	Left	-9	4.5
	$\mathrm{Env}_M(x) = 1 - \mathrm{Env}_{LR}(x)$	x_R	Right	-4.5	9
		β	Rapidness	-4.5	9
Infinite Well	$V(x) = \begin{cases} 0 & \text{if } x_l < x < x_r \\ \infty & \text{if } \textit{otherwise} \end{cases}$	x_l	Left Well	-9	4.5
		x_r	Right Well	-4.5	9
		$x_r - x_l$	Width	1	8
Harmonic	$V(x) = \frac{1}{2}m\omega^2(x-x_0)^2$	ω	Angular Freq.	0.01	3
		x_0	Equilibrium	-5	5
DI Gaussian	$V(x) = -A_1\exp(\frac{(x-\mu_1)^2}{\sigma_1^2}) - A_2\exp(\frac{(x-\mu_2)^2}{\sigma_2^2})$	A_1, A_2	Amplitude	1	10
		μ_1, μ_2	Mean	-5	5
		σ_1, σ_2	Standard Dev.	0.5	4
Random#1	$V(x_{i+1}) = V(x_i) + [X \sim \mathcal{N}(\mu, \sigma)]$	μ	Mean	-4	4
		σ	Standard Dev.	0.5	4
Random#2	Summation of sines and cosines with random coefficients	Number of Terms		1	100
		A_1, A_2	Amplitude	-4	4
		n_1, n_2		-6.30	6.30
		σ		0.1	10
Random#3	Subtraction of two binary grid	Scale Factor		8	8

The second algorithm

Algorithm 2 RandomPotential2

```
1: procedure RandomPotential2
2:     $Nterms = RandomInteger(1, 100)$
3:     for i = 0 to $Nterms$ do
4:         $A = GaussianDistributedRandomNumber()$
5:         $B = GaussianDistributedRandomNumber()$
6:         $n_1 = GaussianDistributedRandomNumber() * \sigma * \pi/width$
7:         $n_2 = GaussianDistributedRandomNumber() * \sigma * \pi/width$
8:         $Potential += A\sin(n_1 x) + B\cos(n_2 x)$
9:     $Potential = GaussianFilter(Potential, \sigma)$
```

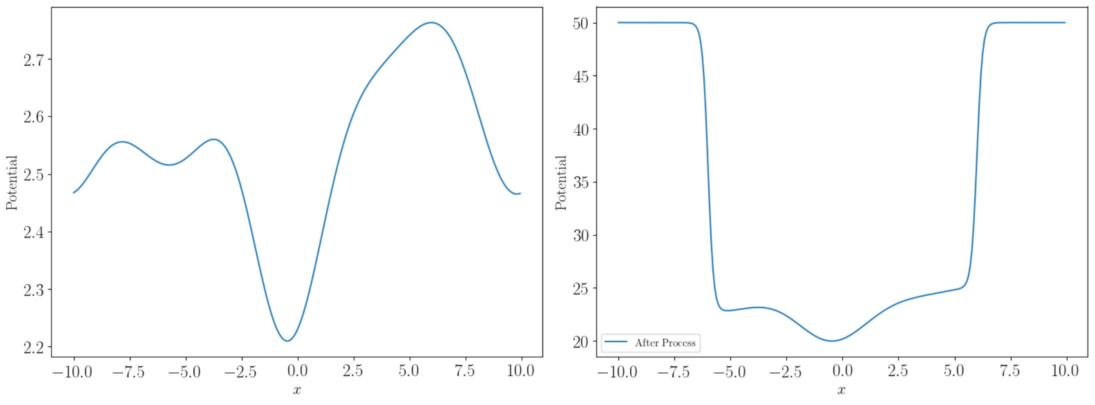


Figure 8: Random Potential 2 Before and After Process

The third one is one dimensional version of the algorithm described in [8] with slight modification. First a binary array of length 16 is generated by assigning random ones or zeros. Then the array is upscaled to 128 by repeating each element of the array 8 times. After that, another binary array of length 8 is generated with the same procedure and upscaled to 64 by repeating elements. Left and right paddings of length 32 is added to the second binary array to be able to do element wise substraction between two binary array. Then, the second array is substracted from the first one to obtain the potential in binary form. Finally, a gaussian filter is applied to make the potential smooth.

Algorithm 3 RandomPotentia3

```
1:  procedure RandomPotential3
2:      $ScaleFactor = 8$
3:      for i = 0 to $NumberOfPoints/ScaleFactor$ do
4:          $BinaryGrid[i] = RandomInteger(0, 1)$
5:      $BinaryGrid = RepeatElements(BinaryGrid, ScaleFactor)$
6:      for i = 0 to $NumberOfPoints/(ScaleFactor * 2)$ do
7:          $BinaryGrid2[i] = RandomInteger(0, 1)$
8:      $BinaryGrid2 = RepeatElements(BinaryGrid, ScaleFactor)$
9:      $Padding = Zeros((Length(BinaryGrid) - Length(BinaryGrid2))/2)$
10:     $BinaryGrid2 = Concatanate(Padding, BinaryGrid2, Padding)$
11:     $Potential = BinaryGrid - BinaryGrid2$
12:     $Potential = GaussianFilter(Potential, \sigma)$
```

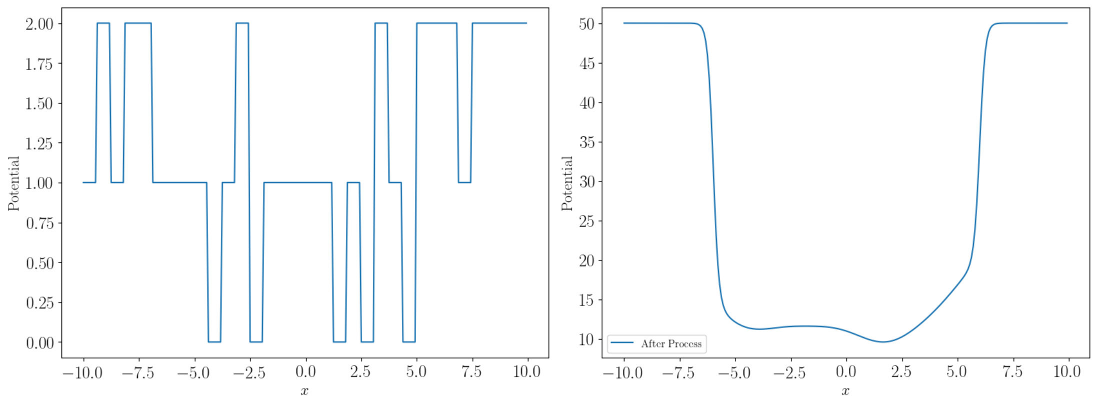


Figure 9: Random Potential 3 Before and After Process

2.3.5 Density and Ground State Energy

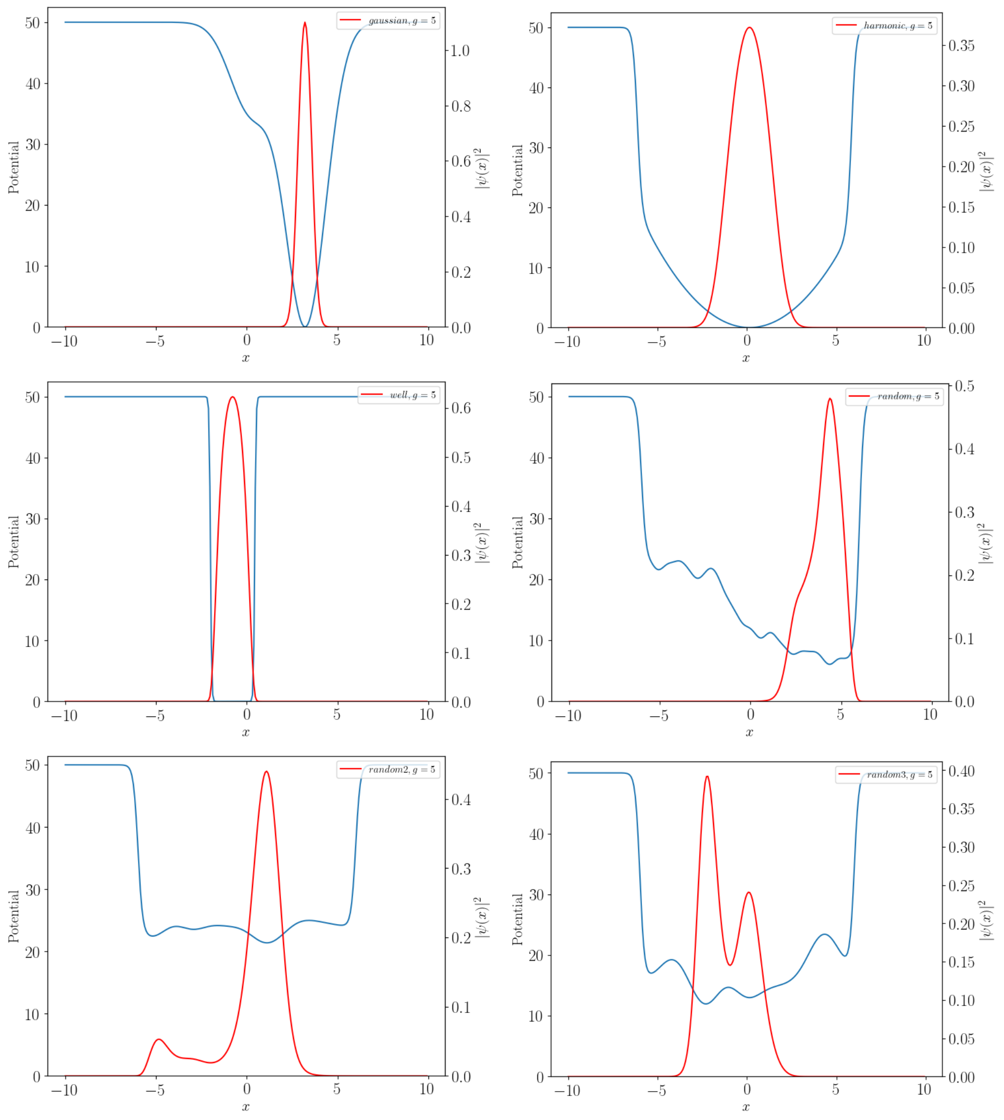


Figure 10: Potentials and Corresponding Ground State Densities

2.3.6 Boundaries. (Table)

2.3.7 Convergence (detailed in APPENDIX)

2.3.8 Dataset generation. (Total number of examples etc)

2.4 Dataset Features

2.4.1 Energy distribution

3 Machine Learning

Our aim is to show that the machine learning methods can be used to bypass GPE equation and physical features of the BEC can be predicted within high accuracy. To do that, we implemented a special type of artificial neural network called convolutional neural network (CNN). It is developed to handle data which have grid-like topology [16] such as an image. Its strength occurs at when the subject has geometrical translation symmetry for instance the shape of a harmonic potential. Since In this section, we briefly introduce notions and mechanisms used in machine learning such as neuron, gradient descent etc. After that, we describe how these are used in our study.

An artificial neural network is made of layers and these layers contain simple units called neurons. These neurons were inspired by the neurons in the human brain. A neuron takes one or more input and generates an output. To do this, it uses its internal variables which are called weights and biases. In general, the number of weights in a neuron is equal to the size of the input and there is one bias value. There is no restriction on the range of weights and biases except computational, they can even take on complex values [17]. Input output relation of a neuron can be described in the following way; if we represent the input of the neuron by $\boldsymbol{x} = [x_1, x_2, x_3, ..., x_n]$ and the function f that describes behavior of the neuron can be written as,

$$f(\boldsymbol{x}, \boldsymbol{\omega}, b) = \sum_{i=1}^{n} \omega_i x_i + b \tag{34}$$

Here, ω_i are the weights and b is the bias. A layer of a network involves neurons that implement with this function. In our example, x_n **inputs represent the potential expression respect to the position** or represents density in the case of inverse problem. The range of the function f is infinite and it can be used directly. However, there are various ways to interpret result of the function f. The most primitive version of interpretation is sending the result of f as an argument to unit step function to generate 0 (False) or 1 (True). In this case,

$$a(f) = u_1(f(\boldsymbol{x}, \boldsymbol{\omega}, b)) \tag{35}$$

where a is called as the activation function of the neuron and it is a unit step function in this case. However, the usage of the step function introduces discontinuity and does not allow to use differential methods to calculate behavior of the neurons when a small change in weights or biases is applied. To handle this problem there are different activation functions. One of the most common activation function is sigmoid function denoted by σ which is continuous version of the step function and it is defined as,

$$\sigma(f) = \frac{1}{1 + e^{-f}} \tag{36}$$

If a neuron defined in this way such that its output given by Eq. (36), then it is called a sigmoid neuron. In the former case which is the step function, the neuron is called as perceptron.

If we denote j^{th} neuron in the l^{th} layer with f_j^l, and the weight from k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the layer l^{th} with ω_{jk}^l, finally the bias of the j^{th} neuron in the l^{th} layer with b_j^l, then, the output of the j^{th} neuron in the l^{th} layer will be;

$$f_j^l(\sigma(f_k^{l-1}), \boldsymbol{\omega}, b) = \sum_{k=1}^{n} \omega_{jk}^l \sigma(f_k^{l-1}) + b_j^l \tag{37}$$

and the activation of the output can be written as;

$$\sigma(f_j^l(\sigma(f_k^{l-1}), \boldsymbol{\omega}, b) \tag{38}$$

Here we use sigmoid function as activation function but it can be any suitable activation function such as hyperbolic tangent $\tanh(f)$ or Rectified Linear Unit (ReLU).

Connection complexity of the neurons may vary. Output of each neuron in one layer can be input of every neuron in next layer. Such networks are called Fully Connected Networks (FCN) or multilayer perceptrons (MLP). If the result of one layer is directly sent to the next layer without any circulation or feedback, then the network is called as Feed Forward Network [1].

We are going to change these weights and biases in such a way that the difference between output generated by network and real value of the corresponding quantity will be minimized. To do that, we are going to use a proxy relation between output and real value which is called as Cost Function [1]. There are different kind of cost functions such as quadratic cost function, cross entropy cost function etc. We are going to use quadratic cost function also called as mean squared error to show the general mechanism and to introduce few notions that directly effects the behavior of the network.

The quadratic cost function is defined as;

$$C(\omega, b) = \frac{1}{2n} \sum_{x} ||\boldsymbol{y} - a(f_L(\boldsymbol{x}))||^2 \tag{39}$$

where n, ω, b are number of examples in the training set, weights and biases, respectively. The subscript L indicates the last layer which is the output, so $f_L(\boldsymbol{x})$ output produced by network and $\boldsymbol{y}$ is real value of the quantity. To minimize C, we can take derivative of the function and can find extremum values, but this method is extremely costly because the total number of neurons in the network is enormous [1]. This problem can be overcame by an iterative algorithm called

Gradient Descent. Since C is also a scalar field the algorithm tries to determine a direction which points to the decrement and moves in that direction with small steps. In each iteration, algorithm repeats itself to reach minimum value.

A small displacement in arbitrary direction which corresponds to a small change in weight or bias or both can be written as;

$$\Delta C = \frac{\partial C}{\partial \omega}\Delta\omega + \frac{\partial C}{\partial b}\Delta b \tag{40}$$

This expression gives information about what happens when small displacement is made. In order to the determine direction of decrement we can rewrite this expression in terms of the gradient since it points to direction of maximum rate of increase. We can define gradient operator as;

$$\boldsymbol{\nabla} C = \left(\frac{\partial C}{\partial \omega}, \frac{\partial C}{\partial b}\right)^T \tag{41}$$

In this case, Eq. (40) can be written as;

$$\Delta C = \boldsymbol{\nabla} C \cdot \Delta \boldsymbol{l} \tag{42}$$

where $\boldsymbol{l} = (\omega, b)^T$. We want to minimize the cost function, thus; in each iteration value of the C must be smaller than the previous one which means that ΔC must be smaller than zero. Therefore, the direction of displacement must be in the opposite direction of the gradient. Rather than calculating the opposite direction for each iteration we can define $\Delta\boldsymbol{l}$ in such a way so that it always points to opposite direction of the gradient.

$$\Delta \boldsymbol{l} = -\eta \boldsymbol{\nabla} C \tag{43}$$

In this case, Eq. (42) becomes;

$$\Delta C = -\eta ||\boldsymbol{\nabla} C||^2 \tag{44}$$

where η is positive definite number and it is called the learning rate. Since it is guaranteed that $||\boldsymbol{\nabla} C||^2 \geq 0$, therefore; $\Delta C \leq 0$. In this situation, learning rate is the step size in each iteration. From Eq. (44) it is intuitive that η can not be a large number because in such a case, the algorithm can miss the minimum point. This situation also brings up the subject that gradient descent does not give guarantee to find the minimum point.

Gradient Descent algorithm has an statistical version called Stochastic Gradient Descent to increase the speed of training process. It is assumed that gradient of m randomly selected examples in the dataset is nearly equal to gradient of the whole dataset and it can be expressed as;

$$\frac{1}{m}\sum_{k=1}^{m}\nabla C_{X_j} \approx \frac{1}{n}\sum_{x=1}^{n}\nabla C_x = \nabla C \tag{45}$$

where ∇C_x is the gradient of a single training input and m is also called as mini-batch size.

It is worth noting that approaching a minimum value is proportional to the number of iterations which equals to number of examples in the dataset, therefore; it is a corollary that there can be two situations[3]. First, the algorithm cannot reach the minimum point due to lack of training examples, and secondly, the algorithm reaches a global or local minimum point and starts to oscillate around it [18]. There are mechanisms to prevent such cases and other optimization problems like direction sensitivity and they are called as adaptive learning rate [8]. In our network we used an algorithm called Adam [19] provided by the framework we used.

The final step in the gradient descent is updating the weights and biases. We use Eq. (44) to perform this task. If we combine Eq. (44) with Eq. (45) and write it in open form, the expression to update the weight and bias can be written as;

$$\begin{aligned} \omega' &= \omega - \frac{\eta}{m}\sum_{j=1}^{m}\frac{\partial C_{X_j}}{\partial \omega} \\ b' &= b - \frac{\eta}{m}\sum_{j=1}^{m}\frac{\partial C_{X_j}}{\partial b} \end{aligned} \tag{46}$$

Numerical calculation of the partial derivatives in the equation is one of the costliest computational operation in the training process. To reduce computational cost, there is a well known algorithm called backpropagation [16]. We use the framework we used it provides similar mechanism called automatic differentiation.

3.1 Machine Learning Framework

We used Pytorch Framework [20] to build our neural networks. It allows the client codes to work on both CPU and GPU via its internal python object called Tensor. If any CUDA supported GPU is available then Pytorch can use GPU without any change in the code. Machine Learning code has three main parts, first one is dataloaders; it reads train and test data from corresponding file and generates tensor dataset object. In this part, manipulation on dataset can

[3]The third case is that algorithm may diverge but we ignore this situation here.

be applied such as normalization, shuffling etc. Second part is implementation of the network. Architecture of the network is represented with a python class inherited from Pytorch's base class for networks called Module. This class also includes a forward method which is responsible for how data will be sent to the next layer. The last part is training and testing. In the training part, the network is iterated with training dataset and loss (result of cost function) is calculated at the end of each iteration. After that, weights and bias are updated accordingly by framework.

3.2 Network architecture

Architecture of the network.
A general figure like in the ML&SE article that describes the work done.
Another figure about internals of the network such as number of layers, how interaction parameter is introduced to the network etc.
Hyperparameters.

3.3 Training

Detailed info about dataset (energy distribution etc).
Indicate that if there is any method to increase the number of examples in low and high energy values.

3.4 Results

4 Inverse Problem

5 Conclusion and Discussion

5.1 Conclusion

5.2 Discussion

5.3 Effects of random potential generation method

5.4 Are there problems in low and high energies compared to the mean

5.5 Inverse problem

References

[1] M. A. Nielsen, “Neural networks and deep learning,” 2015.

[2] PhysicsML, “Papers.”

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

[4] Z. Cai, “Approximating quantum many-body wave-functions using artificial neural networks,” *arXiv preprint arXiv:1704.05148*, 2017.

[5] L. Wang, “Discovering phase transitions with unsupervised learning,” *Physical Review B*, vol. 94, no. 19, p. 195105, 2016.

[6] E. M. Stoudenmire and D. J. Schwab, “Supervised learning with quantum-inspired tensor networks,” *arXiv preprint arXiv:1605.05775*, 2016.

[7] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, “Quantum machine learning, 2016,” *arXiv preprint arXiv:1611.09347*.

[8] K. Mills, M. Spanner, and I. Tamblyn, “Deep learning and the schrödinger equation,” *Physical Review A*, vol. 96, no. 4, p. 042113, 2017.

[9] C. J. Pethick and H. Smith, *Bose-Einstein condensation in dilute gases*. Cambridge university press, 2002.

[10] J. Rogel-Salazar, “The gross–pitaevskii equation and bose–einstein condensates,” *European Journal of Physics*, vol. 34, no. 2, p. 247, 2013.

[11] L. Pitaevskii and S. Stringari, *Bose-Einstein condensation and superfluidity*, vol. 164. Oxford University Press, 2016.

[12] C. F. Barenghi and N. G. Parker, *A primer on quantum fluids*. No. arXiv: 1605.09580, Springer, 2016.

[13] H.-L. Zheng and Q. Gu, “Dynamics of bose-einstein condensates in a one-dimensional optical lattice with double-well potential,” *Frontiers of Physics*, vol. 8, no. 4, pp. 375–380, 2013.

[14] G. R. Dennis, J. J. Hope, and M. T. Johnsson, “Xmds2: Fast, scalable simulation of coupled stochastic partial differential equations,” *Computer Physics Communications*, vol. 184, no. 1, pp. 201–208, 2013.

[15] X. Antoine and R. Duboscq, "Gpelab, a matlab toolbox to solve gross–pitaevskii equations i: Computation of stationary solutions," *Computer Physics Communications*, vol. 185, no. 11, pp. 2969–2991, 2014.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[17] H.-G. Zimmermann, A. Minin, and V. Kusherbaeva, "Comparison of the complex valued and real valued neural networks trained with gradient descent and random search algorithms.," in *ESANN*, 2011.

[18] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[19] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

A APPENDIX A

Obtaining harmonic trap potential scaling from Eq. (30)

$$V(x) \to V(\widetilde{x}) \to \widetilde{V}(\widetilde{x}) \tag{47}$$

$$\widetilde{V}(x) \equiv \frac{V(x)}{\gamma E_0} \tag{48}$$

$$\widetilde{x} \equiv \frac{x}{\beta L} \tag{49}$$

$$V(x) = \frac{1}{2} m\omega^2 (x - z_0)^2 \tag{50}$$

$$V(\widetilde{x}) = \frac{1}{2} m\omega^2 \beta^2 L^2 (\widetilde{x} - \widetilde{z}_0)^2 \tag{51}$$

$$\widetilde{V}(\widetilde{x}) = \frac{1}{2} m\omega^2 \frac{\beta^2 L^2}{\gamma E_0} (\widetilde{x} - \widetilde{z}_0)^2 \tag{52}$$

The coefficient of this equation must be dimensionless, therefore;

$$\frac{1}{2} m\omega^2 \frac{\beta^2 L^2}{\gamma E_0} = C \tag{53}$$

Where C is a positive constant. We know that $E_0 = \frac{\hbar^2}{2m}$ and the definition of α is given as,

$$\frac{\hbar^2}{2m\gamma E_0}\frac{1}{\beta^2 L^2} = \alpha \tag{54}$$

thus;

$$\frac{1}{\gamma} = \alpha\beta^2 L^2 \tag{55}$$

if we plug Eq. (55) in to Eq. (53), then equation becomes,

$$\frac{m^2\omega^2}{\hbar^2}\alpha\beta^4 L^4 = C \tag{56}$$

in the case of expressing same physical system, m and ω must be constant. In that case $\alpha = C(\beta L)^{-4}$.

In this case, α becomes

$$\alpha = \frac{1}{2}\left(\frac{\hbar\omega}{\gamma E_0}\right)^2$$

Conventionally, α is set to 1/2, therefore;

$$\hbar\omega = \gamma E_0$$

$$\beta L = \sqrt{\frac{\hbar}{m\omega}}$$

βL is generally defined as harmonic oscillator length ℓ

$$\ell = \sqrt{\frac{\hbar}{m\omega}}$$

$$\widetilde{\mu} = \frac{\mu}{\hbar\omega}$$

$$\widetilde{g} = \frac{g}{\hbar\omega\ell}$$

Finally, dimensionless GPE scaled for harmonic trapping potential can be written as,

$$\widetilde{\mu}\widetilde{\psi} = -\frac{1}{2}\frac{d^2\widetilde{\psi}}{d\widetilde{x}^2} + \frac{1}{2}\widetilde{x}^2\widetilde{\psi} + \widetilde{g}|\widetilde{\psi}|^2\widetilde{\psi}$$