Summary

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

1 Introduction

2 Gross Pitaevskii Equation

2.1 Types of Potentials

2.2 Analytical Solution and Approximation

2.3 Numerical Solution and XMDS Framework

3 Problem Statements and Dataset Generation

GPE can be solved with briefly described methods above. These methods must be applied every time when there is a change in the equation such as different trapping potential or parameters. For example, when interaction parameter is changed, if there is no analytic solution, numerical method must be reapplied. With machine learning these steps can be bypassed and information about the system can be obtained instantly with only one time cost which is training process of the network. **REF**. Here **REF**, authors managed to apply deep learning method to the Schrodinger Equation to obtain ground state energy of the system under different potentials.

We try to built a neural network to predict ground state energy of a BEC for a given harmonic trapping potential and interaction parameter. We also try inverse problem which is prediction of interaction parameter for a given potential and density function.

An artificial neural network is made of layers and these layers contains simple units called neurons. These neurons can be thought as primitive version of 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 weight and bias. In general, number of weights in a neuron is equal to the input size of the

neuron. Input output relation of a neuron can be described with the following way.

Let $\boldsymbol{x} = [x_1, x_2, x_3, ..., x_n]$ be input of the neuron and f be the function that describes behavior of the neuron.

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

Here, ω_i is weight and b is bias. A layer of a network involves neurons behaves like this function. In our example, x_n inputs represent the potential expression respect to the position or density function. **FIGURE OF NETWORK and explanation**.

If we denote j^{th} neuron in the l^{th} layer with f_j^l, then, connection between two neuron will be;

$$f_s^{l+1}(f_j^l(\boldsymbol{x}, \boldsymbol{\omega}, b), \boldsymbol{\omega}, b) = \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} (x_j \omega_j + b_1)\omega_i + b_2 \tag{2}$$

$$\tag{3}$$

Connection complexity of the neurons may vary. Output of each neuron in one layer can be input of a neuron in next layer. Such a networks are called Fully Connected Networks. **REF**. If result of one layer directly is sent to the next layer without any circulation or feedback, then the network is called as Feed Forward Network. **REF**

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. **REF**. There are many 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 general mechanism and to introduce few notions that directly effects the behavior of the network.

Quadratic cost function is defined as;

$$C(\omega, b) = \frac{1}{2n} \sum_{x} ||\boldsymbol{a} - f_L(\boldsymbol{x})||^2 \tag{4}$$

where ω, b are weights and biases respectively. $f_L(\boldsymbol{x})$ output produced by network and $\boldsymbol{a}$ is real value of the quantity. To minimize C, we can take the 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.

REF. This problem is overcame by an iterative algorithm called Gradient Descent. Since C is also a scalar field the algorithm tries to determine a direction to which points the decrement and moves to that direction with small steps. In each iteration, algorithm repeats itself to reach minimum value.

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

$$\Delta C = \frac{\partial C}{\partial \omega}\Delta\omega + \frac{\partial C}{\partial b}\Delta b \tag{5}$$

This expression gives information about what happens when small displacement is done but it does not give any information about direction of decrement. To determine direction of decrement we can rewrite this expression in terms of gradient since it is by definition points to direction of maximum rate of increase.

We can define gradient operator as;

$$\boldsymbol{\nabla} C = \left(\frac{\partial C}{\partial \omega}, \frac{\partial C}{\partial b}\right)^T \tag{6}$$

In this case, Eq. **??** can be written as;

$$\Delta C = \boldsymbol{\nabla} C . \Delta \boldsymbol{l} \tag{7}$$

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

$$\Delta \boldsymbol{l} = -\eta \boldsymbol{\nabla} C \tag{8}$$

In this case, Eq. 7 becomes;

$$\Delta C = -\eta ||\boldsymbol{\nabla} C||^2 \tag{9}$$

where η is positive definite number and it is called as 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. 9 it is intuitive that η can not be a large number because in such a case, one can miss the minimum point. **FIGURE**. This situation also brings up the subject that gradient descent does not give guarantee to find the minimum point.**REF** If there is no enough example to iterate gradient descent, cost function may be lower than its first condition but

it will not be minimized. **It may even retain its first condition ex:lr=0.05**

3.1 Dataset and Dataset Generation

As mentioned, we solved NLSE for four different interaction parameters, therefore; there are four different corresponding datasets and each of them contains 10.000 elements. We used 8500 of them as training examples and 1500 of them for test. An element of a dataset involves an array containing potential values respect to the position and an another array containing interaction, kinetic, potential and total energy values respectively.

The array containing potential energy is generated according to harmonic trap expression in XMDS, thus; **randomness** of the dataset occurs in angular frequency and shift of equilibrium point. We supplied these random angular frequency and shift values to the XMDS within predetermined limits. Angular frequency can take values between 0.5 and 2. Shift of the equilibrium point is determined due to boundary conditions since **density function** must be zero at boundaries. In numerical solution of the NLSE, domain of the **transverse dimension** is between -10 to 10. Taking the maximum magnitude of the shift as ∓ 5 enough to ensure that density function goes to zero at infinity.

4 Machine Learning for NLSE

We used Pytorch Framework to build our neural networks. It allows the client codes 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. Code have three main parts, first one is dataloaders; it reads train and test data for specified interaction parameter from corresponding file and generates tensor dataset object. **continue**

We implemented two different types of neural network; feed forward (FNN) and convolutional neural network (CNN).

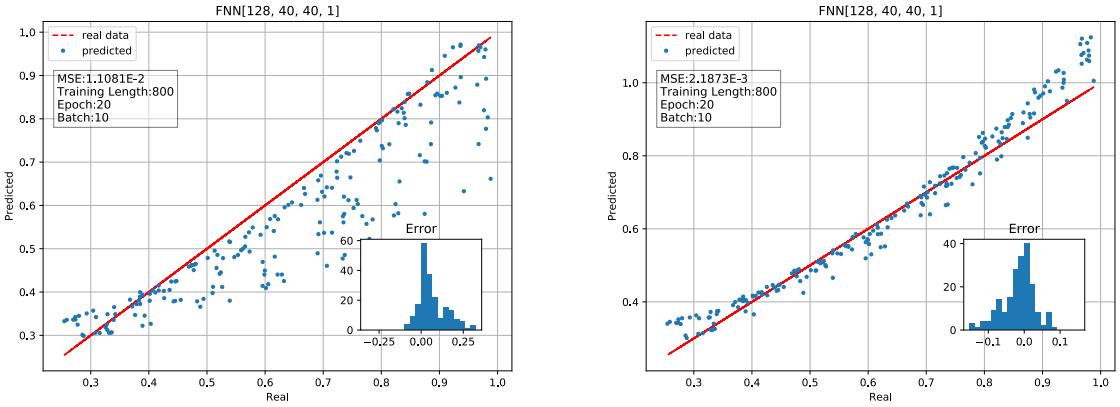
4.1 Architecture

FNN involves 128 input neurons as input layer, next layer is first hidden layer with 30 neurons, the second is same as first hidden layer, the next one is last hidden layer with 10 neurons and the last layer is output layer. Totally there are 5 layers in our FFN and it will be denoted as $FNN[128, 30, 30, 10, 4]$. Rectified linear unit (ReLU) is used for each forward except output. No operation is applied to the output neuron. Learning rate of the FNN is fixed and it is 0.001. Cost function is mean squared error (MSE) and optimization is done with Adam.

CNN has two convolution layers, two maxpool layers and three fully connected layer and last layer of the fully connected part is output layer. Maxpooling is applied to output of the first and second convolution layers. ReLu is also applied each forward except output neuron same as in the FNN. Fully connected part of the CNN is $FNN[310, 100, 20, 1]$. Learning rate, cost function and optimizer of the CNN is same as FNN which are 0.001, MSE and Adam respectively.

4.2 Hyperparameters

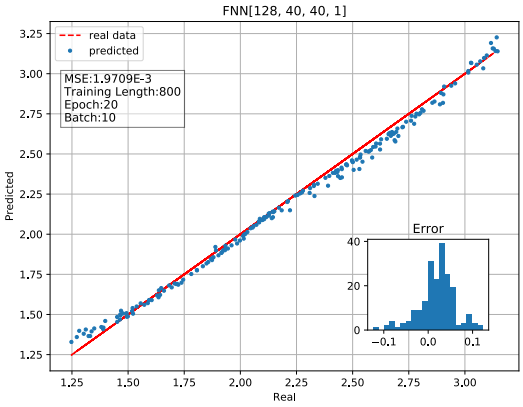
We firstly started with smaller dataset which involves 800 elements for trainig 200 for test to see response of the architecture and to obtain a range for learning rate.



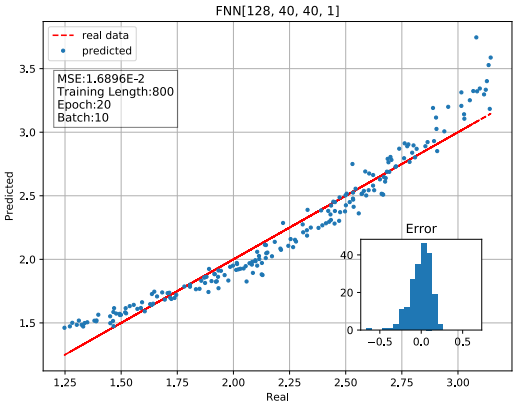
(a) FNN[128, 40, 40, 1], lr = 0.003

(b) FNN[128, 40, 40, 1], lr = 0.001

Figure 1: In this example interaction parameter g, is set to zero. Even learning rate 0.001 seems more accurate, it is trivial to expect that change in interaction parameter will effect the result.



(a) FNN[128, 40, 40, 1], lr = 0.003

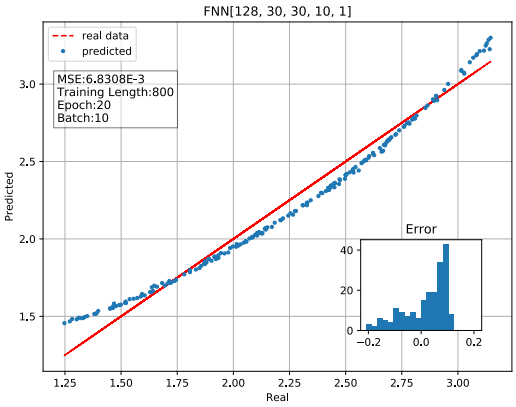


(b) FNN[128, 40, 40, 1], lr = 0.001

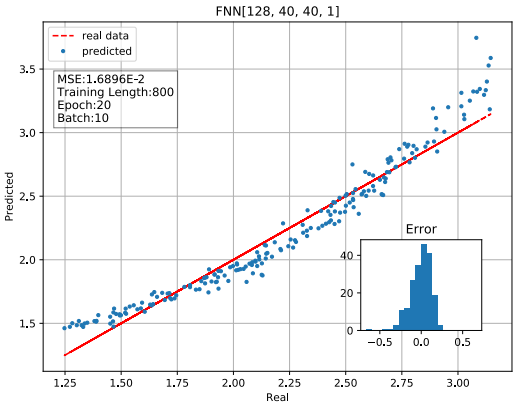
Figure 2: Here, interaction parameter, g is 10. Predictions of the network with learning rate 0.003 more accurate than 0.001.

comment about range of the learning rate ?

First architecture we used had only 4 layers and it was not sufficient.



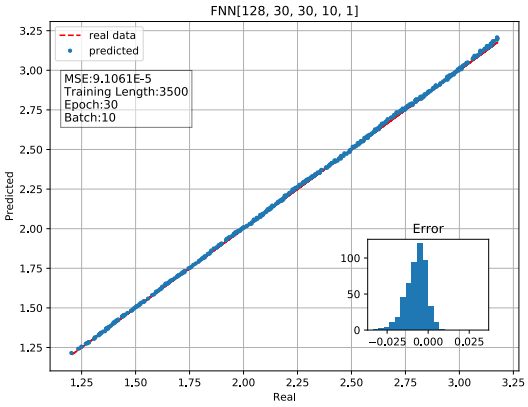
(a) FNN[128, 30, 30, 10, 1]



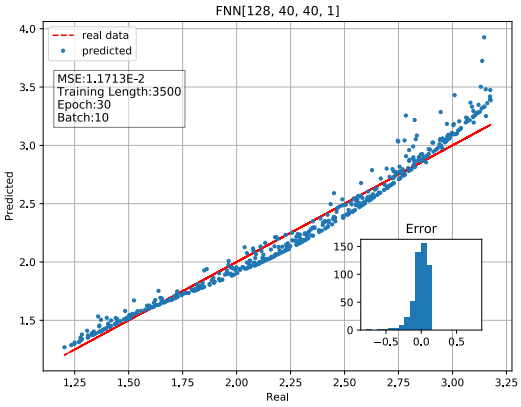
(b) FNN[128, 40, 40, 1]

Figure 3: Here interaction paramater g, is again set to 10. Batch size is 10 and total number of epoch is 20 for this dataset.

Then we increased the number of layers to 5 and tried different number of neuron combinations. Each of these tried network behaved different for different g values. For example, MSE of the FNN[128, 40, 40, 20, 1] for g = 10 is $5.11x10^{-3}$ but for g = 0 it is $2.00x10^{-3}$



(a) FNN[128, 30, 30, 10, 1]



(b) FNN[128, 40, 40, 1]

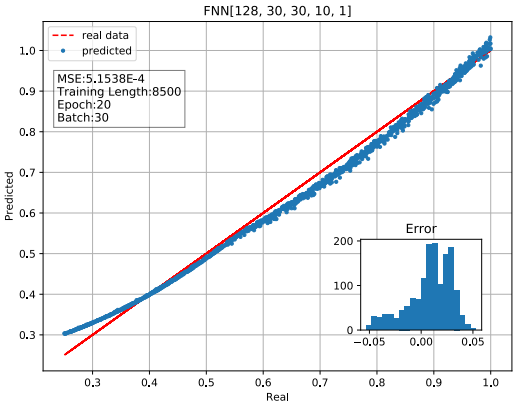
Figure 4: 3500/500

After narrowing the number of candidate architectures with small dataset, we increased the size of the dataset to 3500/500 to get more information about distribution of the predictions and to determine other two hyperparameters; mini batch size and epoch. **Add results of this arch**.

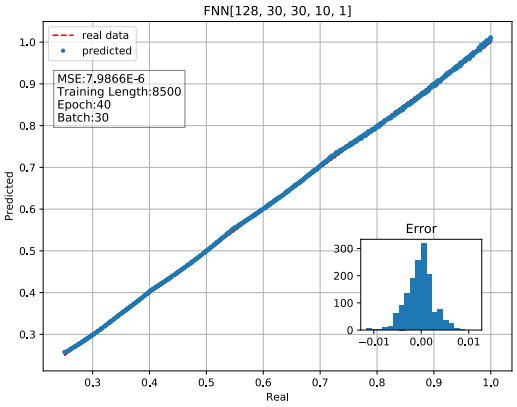
4.3 Training Results

4.3.1 Non-interacting System

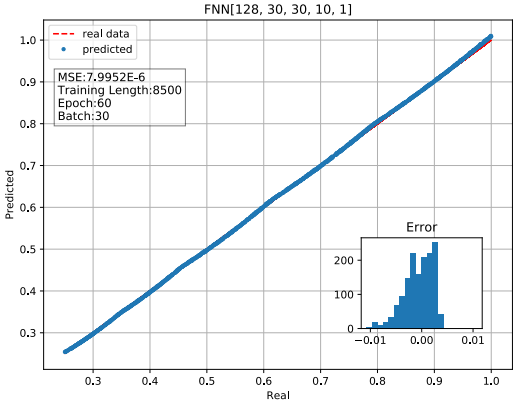
In non-interacting system, GPE reduces to SE which does not involve non-linear term.



(a) FNN[128, 30, 30, 10, 1]



(b) FNN[128, 30, 30, 10, 1]



(c) FNN[128, 40, 40, 1]

Figure 5: Since the system has no interaction parameter, problem is relatively easy when it is compared to systems that involve interaction. In (a) it is obvious network requires more training example and it reaches a satisfactory level in (b). Supplying 20 more epochs nearly does not effect the prediction accuracy.

asdasd asda

4.3.2 Interacting Systems