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 Schrodinger 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

A Bose-Einstein Condensate (BEC) is described by Gross Pitaevskii Equation (GPE) also known as non-linear Schrodinger Equation. General form of GPE is given as,

$$i\hbar\frac{\partial\Psi}{\partial t} = \frac{-\hbar}{2m}\nabla^2\Psi + V(\boldsymbol{r},t)\Psi + g|\Psi|^2\ \Psi \tag{1}$$

where where $\hbar$ is Planck constant, $\boldsymbol{r}$ is position vector, t is time, Ψ is wave function, m is mass, ∇^2 is Laplacian, V is potential, g is interaction parameter and it is defined as

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

where a_s s wave scattering length. Non-linearity of GPE comes from interaction parameter. If there is no interaction GPE reduces to the Schrodinger Equation (SE) and becomes linear. By definition, g can be positive or negative. If $g > 0$, it represents repulsive interaction, and if $g < 0$, it means interaction is attractive. **REF**.

In general, potential V is time independent harmonic trapping potential **REF**. In such a case, the potential is only function of position $V(\boldsymbol{r})$ and it can be given as,

$$V(x,y,z) = \frac{1}{2}m(\omega_x^2x^2 + \omega_y^2y^2 + \omega_z^2z^2) \tag{3}$$

where ω is angular frequency. Then separation of variables can be applied and it can be obtained that there is a solution in stationary form. In this case factorized wave function is,

$$\Psi(\boldsymbol{r}) = \psi(\boldsymbol{r})e^{-i\mu t/\hbar} \tag{4}$$

Here, μ is chemical potential. Time independent GPE is then,

$$\mu\psi = \frac{-\hbar^2}{2m}\nabla^2\psi + V(\boldsymbol{r})\psi + g|\psi|^2\psi \tag{5}$$

BEC system can also be studied in one dimension such that making angular frequencies $\omega_x, \omega_y \gg \omega_z$ and keeping their energy order much greater than condensate's energy $\hbar(\omega_x\omega_y)^{1/2} \gg \mu$ enables us to confine the dynamics of the system in one dimension and it can be described with corresponding 1D GPE equation. **REF**. To obtain 1D GPE one can rewrite wave function,

$$\psi(x, y, z, t) = \psi_z(z, t)\psi_0(x)\psi_0(y) \tag{6}$$

continue

$$\mu'\psi_z = \frac{-\hbar^2}{2m}\frac{d^2\psi_z}{dz^2} + \frac{1}{2}m\omega_z^2 z^2\psi_z + g'|\psi_z|^2\psi_z \tag{7}$$

where μ' and g' one dimensional effective chemical potential and interaction strength respectively and they are given as,

$$\mu' = \mu - \frac{\hbar}{2}(\omega_x - \omega_y), \quad g' = \frac{g}{2\pi l_x l_y} \tag{8}$$

Dimensionless GPE

It is convenient to work with dimensionless quantities, to make equation dimensionless one can define $z = a_0\widetilde{z}$. In this case Eq. 7 becomes,

$$\mu\psi = \frac{-\hbar^2}{2ma_0^2}\frac{d^2\psi}{d\widetilde{z}^2} + \frac{1}{2}m\omega^2 a_0^2\widetilde{z}^2\psi + g|\psi|^2\psi \tag{9}$$

(Here we omit scripts). After that, setting $\hbar\omega = \hbar^2/ma_0^2$, and plugging a_0^2 in to the Eq. 9,

$$\mu\psi = -\frac{1}{2}m\omega^2\frac{d^2\psi}{d\widetilde{z}^2} + \frac{\hbar^2}{2m}\widetilde{z}^2\psi + g|\psi|^2\psi \tag{10}$$

If we divide Eq. 10 to $\hbar\omega$ and define $\widetilde{\mu} = \mu/\hbar\omega$,

$$\widetilde{\mu}\psi = -\frac{1}{2\sqrt{a_0}}\frac{d^2\psi}{d\widetilde{z}^2} + \frac{1}{2\sqrt{a_0}}\widetilde{z}^2\psi + \frac{g|\psi|^2\psi}{\hbar\omega} \tag{11}$$

From normalization condition,

$$\int |\psi|^2 dz = \int |\psi|^2 a_0 d\widetilde{z} = N \tag{12}$$

we can define $\widetilde{\psi} = (\sqrt{a_0}/\sqrt{N})\psi$. The third term on the right in Eq. 11

becomes,

$$\frac{g|\psi|^2}{\hbar\omega} = \frac{ga_0}{N\hbar\omega}|\widetilde{\psi}|^2 \tag{13}$$

Finally defining,

$$\frac{ga_0}{N\hbar\omega_z} = \widetilde{g} \tag{14}$$

dimensionless GPE can be written as,

$$\widetilde{\mu}\widetilde{\psi} = -\frac{1}{2}\frac{d^2\widetilde{\psi}}{d\widetilde{z}^2} + \frac{1}{2}\widetilde{z}^2\widetilde{\psi} + \widetilde{g}|\widetilde{\psi}|^2\widetilde{\psi} \tag{15}$$

2.1 Types of Potentials

2.2 Analytical Solution and Approximation

There is no general analytical solution of GPE for harmonic trapping potential except the case which interaction parameter is zero. In this case, GPE reduces to the SE and solution is well known.

2.3 Numerical Solution and XMDS Framework

Analytical solution of GPE is known for only few cases such as uniform condensate which is $V = 0$. Most of the time GPE is solved with numerical techniques or approximations. In this study, we used a framework called XMDS, implemented specifically to solve differential equation systems with well optimized numerical methods. Equation system 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 pre-specified numerical method. In our program, XMDS takes angular frequency, shift of equilibrium point, value of interaction parameter and number of particles in the system externally. It generates trapping potential and other quantities internally with supplied expressions.

Cross-check of the framework has done in two ways. First, since there exists an analytical solution for $g = 0$, we compared the results of the XMDS with analytical ones, and we also used the program here **REF**, and compared few results for situations where analytical solution does not exist.

3 Problem Statements and Dataset Generation

GPE can be solved with stated 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 artificial 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 and there is only one bias value per neuron. There is no restriction to range of weights and biases except computational, they can even take complex values **REF**. 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.

$$f(\boldsymbol{x}, \boldsymbol{\omega}, b) = \sum_{i=1}^{n} x_i \omega_i + b \tag{16}$$

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**. Notice that, the range of the function f is infinite. There are various ways to interpret result of function f. The most primitive version of interpretation is sending the result of f as a argument to unit step function. In this case,

$$a = u_1(f(\boldsymbol{x}, \boldsymbol{\omega}, b)) \tag{17}$$

where a is called as activation value of neuron and u_1 is the activation function. However, usage of the step function makes the network discontinuous and does not allow to use differential methods to calculate behavior of the neurons when a small change in weights or biases is applied. One of the most common activation

function is sigma function and it is defined as,

$$\sigma(f) = \frac{1}{1 + e^{-f}} \tag{18}$$

If a neuron defined with this way such that its output given by Eq. 18, then it is called as sigmoid neuron. In former case which is step function, then, 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} 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 connection between two neuron in sequential layers will be;

$$f_s^{l+1}(\sigma(f_j^l(\boldsymbol{x}, \boldsymbol{\omega}, b)), \boldsymbol{\omega}, b) = \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} (x_j \omega_j + b_1)\omega_i + b_2 \tag{19}$$

Eq requires correction

Here we used σ as activation function but it can be any suitable activation function such as hyperbolic tangent $tanh(f)$.

Connection complexity of the neurons may vary. Output of each neuron in one layer can be input of every neuron in next layer. Such a networks are called Fully Connected Networks. **REF**. If 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. **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;

$$C(\omega, b) = \frac{1}{2n} \sum_x ||\boldsymbol{a} - f_L(\boldsymbol{x})||^2 \tag{20}$$

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;

$$\Delta C = \frac{\partial C}{\partial \omega}\Delta\omega + \frac{\partial C}{\partial b}\Delta b \tag{21}$$

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;

$$\boldsymbol{\nabla} C = \left(\frac{\partial C}{\partial \omega}, \frac{\partial C}{\partial b}\right)^T \tag{22}$$

In this case, Eq. 21 can be written as;

$$\Delta C = \boldsymbol{\nabla} C . \Delta \boldsymbol{l} \tag{23}$$

where $\boldsymbol{l} = (\omega, b)^T$. What we want again is to 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.

$$\Delta \boldsymbol{l} = -\eta \boldsymbol{\nabla} C \tag{24}$$

In this case, Eq. 23 becomes;

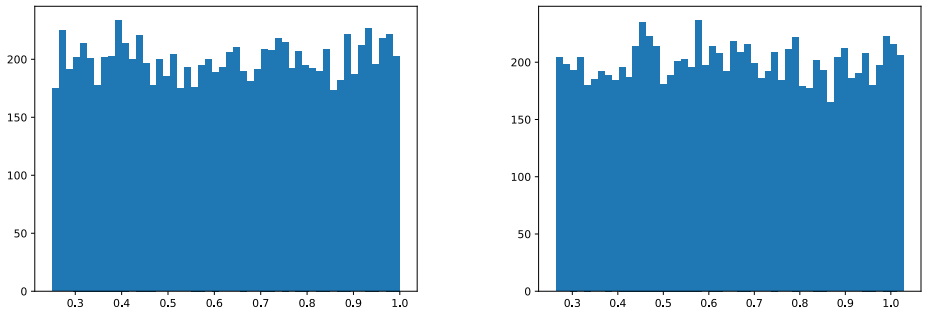
$$\Delta C = -\eta ||\boldsymbol{\nabla} C||^2 \tag{25}$$

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. 25 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**.

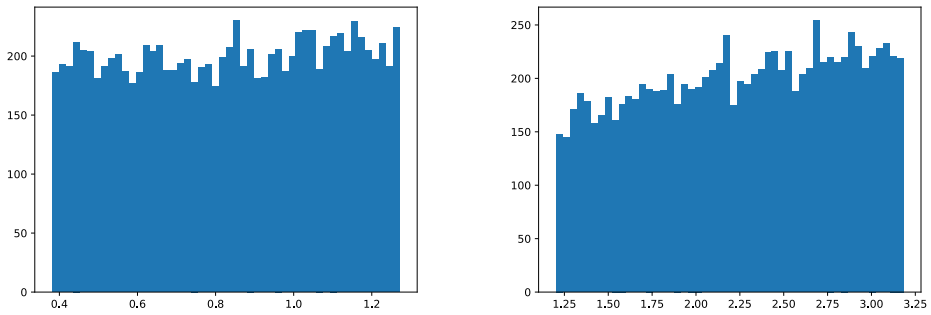
3.1 Dataset and Dataset Generation

As mentioned, we solved GPE 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 if otherwise is specified. 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. (Representation of the energy values may vary, because we first tried to predict total energy values. In this case, the array containing all types of energy values only involves total energy).

TABLE of parameters



(a) g = 0 (b) g = 0.1



(c) g = 1 (d) g = 10

Figure 1: Here histograms represents total energy distributions for different interaction parameter values.

The array containing potential energy is generated according to harmonic trap expression in XMDS, thus; we only supplied angular frequency and shift of equilibrium to the XMDS randomly 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 GPE, domain of the z 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. In both angular frequency and shift, we used uniformly generated random numbers. Total energy distribution is given in Fig 1. Another type distribution is not required in our problem since the potential type is fixed but there are examples such as here **REF**, which different distribution is used.

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. In this part, manipulation on dataset can 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 involves forward method which is responsible for how data will be sent to the next layer.

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

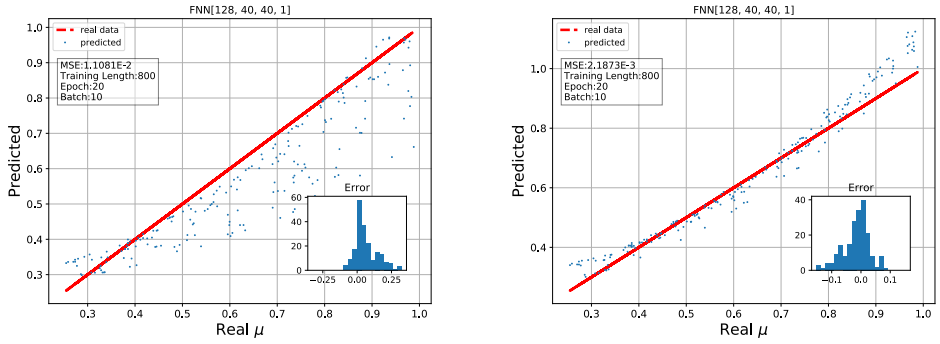
4.1 Architecture

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.

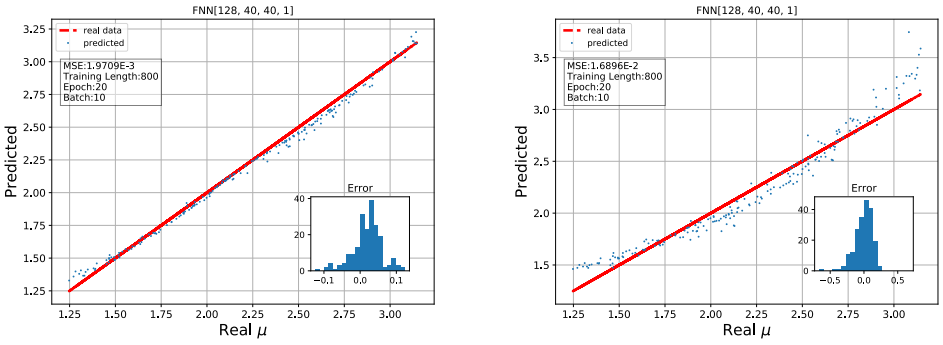
4.2 Hyperparameters

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. The size of this dataset is **very small** but it can be give an idea about the range of learning rate since stochastic gradient descent is an statistical approach. **REF**. We first set learning rate 3 and decrease it, until 0.01 prediction of the network was extremely poor. For lower values, predictions were more encouraging. In this section, we give you the effect of learning rate for two different architectures and interaction parameters.



(a) FNN[128, 40, 40, 1], $\eta = 0.003$ (b) FNN[128, 40, 40, 1], $\eta = 0.001$

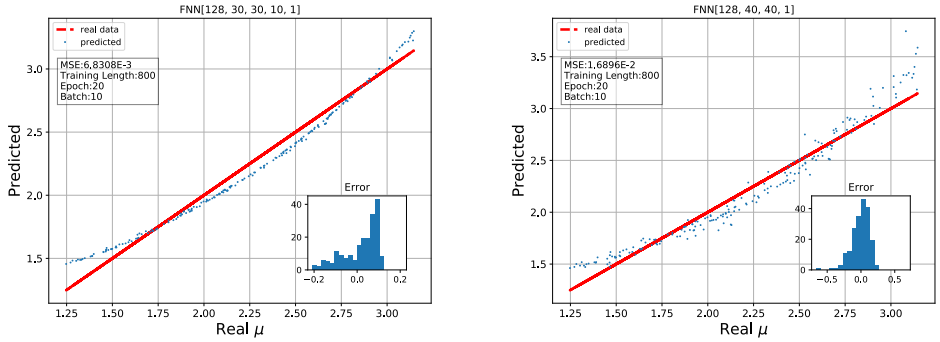
Figure 2: Here both figures represent True Energy of Ground State (unit) vs Predicted Ground State Energy (unit). Their hyperparameters are identical except learning rate. Total number of epoch is 20 and batch size is 10. In this example interaction parameter g, is zero. Even learning rate 0.001 seems more precise, 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

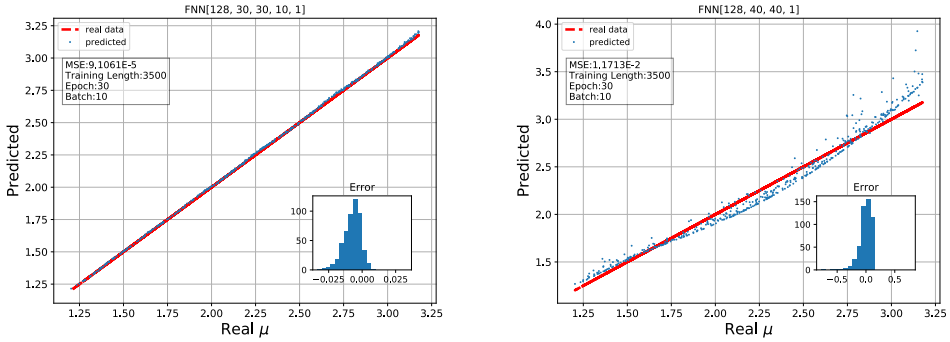
Figure 3: Here, interaction parameter, g is 10. It is clear that precision of the network with learning rate 0.003 is higher than 0.001. Of course, dramatic increase in the training dataset length may affect them both to converge same precision but our intention here to show that small change in learning rate causes different results.

After determination of learning rate we added extra one hidden layer to the network and tried some different number of neurons combination in the layers.



(a) FNN[128, 30, 30, 10, 1] (b) FNN[128, 40, 40, 1]

Figure 4: Here interaction paramater g, is again 10. Batch size is 10 and total number of epoch is 20 for this example. Precision of the network of 5 layer (a) is higher than 4 layer (b). Bias in (a) can be eliminated by increasing training dataset length.



(a) FNN[128, 30, 30, 10, 1] (b) FNN[128, 40, 40, 1]

Figure 5: Total training dataset length is increased to 3500 and test dataset length is 500. Batch size, Learning rate are same as example given in Fig. 4 but the epoch is 30. The bias is eliminated. Precision and accuracy of the network of 5 layers is sufficient.

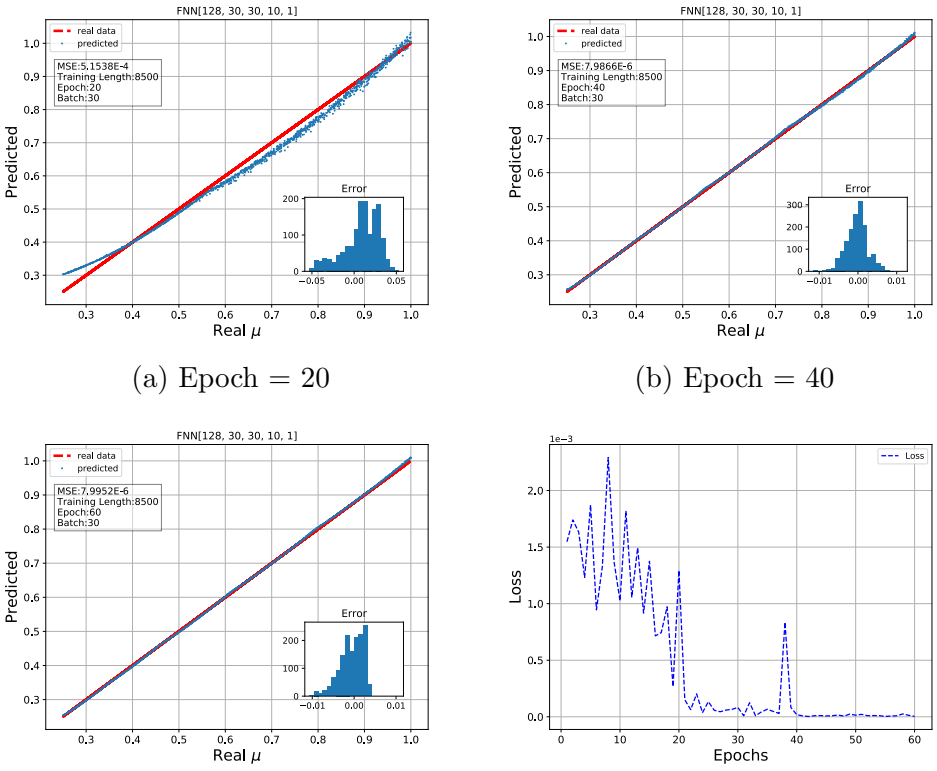
In conclusion, we chose learning rate η as 0.001 and started to training process. The other two hyperparameters epoch and mini batch size determined in training process such that if increase in total number of epochs improves the results dramatically then new epoch value is this one and the mini batch size is determined with the same way. These steps are done for both FFN and CNN. The total number of epochs is 60, and mini batch size is 30.

4.3 Training Results

We give the result of total energy predictions per 20 epochs for both FFN and CNN also to show that how predictions change. Separate energy predictions are given directly. In inverse problem three different scenarios are presented. All trainings are done with same program and the architectures are identical, only the training and test datasets are changed.

4.3.1 Non-interacting System

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



(a) Epoch = 20

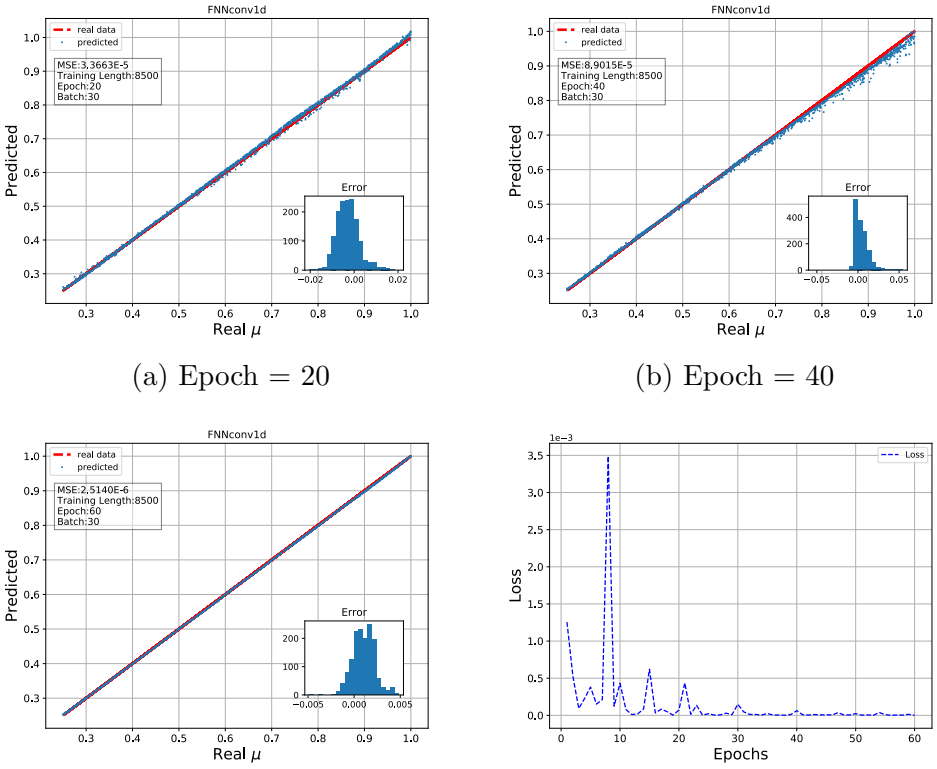
(b) Epoch = 40

(c) Epoch = 60

(d) Loss

Figure 6: In figures, x axis represents true dimensionless total energy values, and y is predicted energy by trained network.

Since the system has no interaction parameter, problem is relatively easy when it is compared to systems that involve interaction. In Fig. 6(a) it is obvious network requires more training example and it reaches a satisfactory level in Fig. 6(b). Supplying another 20 more epochs nearly does not effect the prediction accuracy.



(a) Epoch = 20

(b) Epoch = 40

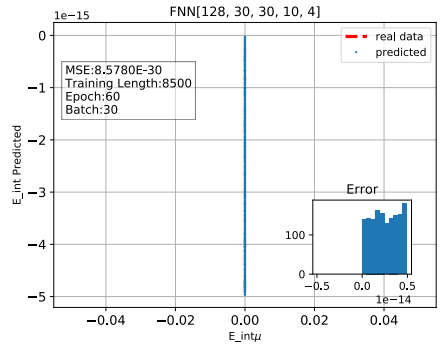
(c) Epoch = 60

(d) Loss

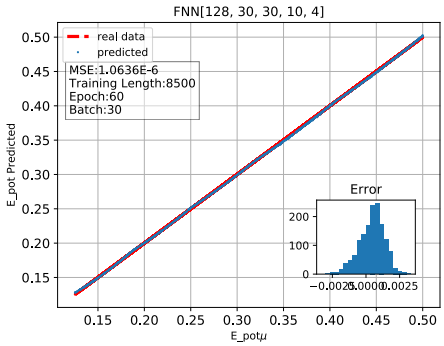
Figure 7: CNN results.

The bias in Fig. 6(a) does not appear in Fig. 7(a). The reason why there is less or no bias in CNN predictions is because CNN's have the ability to handle translations in dataset. In our dataset, translation corresponds to shift of equilibrium point.

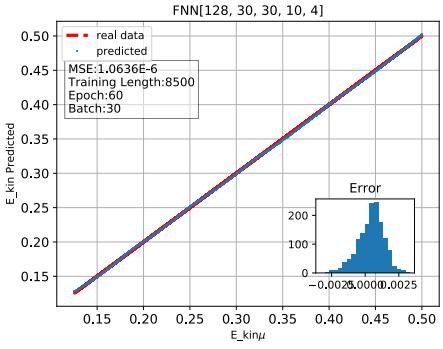
Accuracy of the CNN is higher than FFN in first 20 epochs. In (a) there is no bias as in Fig. 6



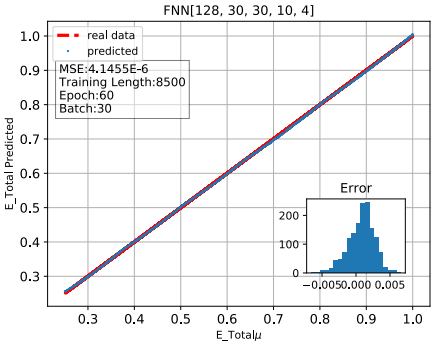
(a) FNN[128, 30, 30, 10, 4]



(b) FNN[128, 30, 30, 10, 4]



(c) FNN[128, 30, 30, 10, 4]

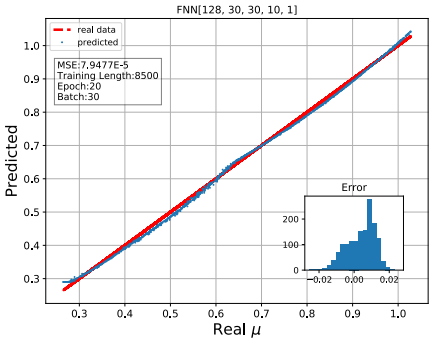


(d) FNN[128, 30, 30, 10, 4]

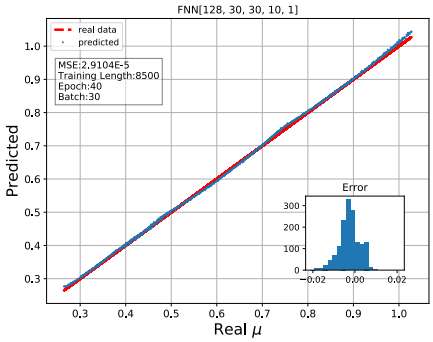
Figure 8: Here (a) is interaction energy, (b) potential energy, (c) kinetic energy and (d) total energy predictions. Only number of neurons in the output layer increased to 4, and still network predicts total energy of the system within same sensitivity as former FNN network.

4.3.2 Interacting Systems

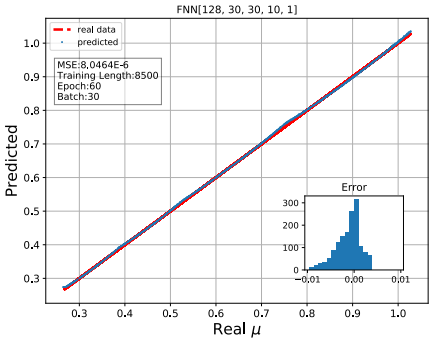
In interacting systems values of interaction parameter g are 0.1, 1, 10, 20 and the results are given in this order.



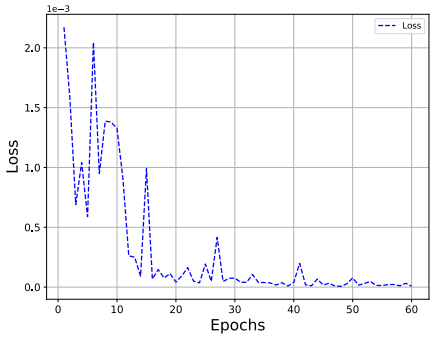
(a) FNN[128, 30, 30, 10, 4]



(b) FNN[128, 30, 30, 10, 4]

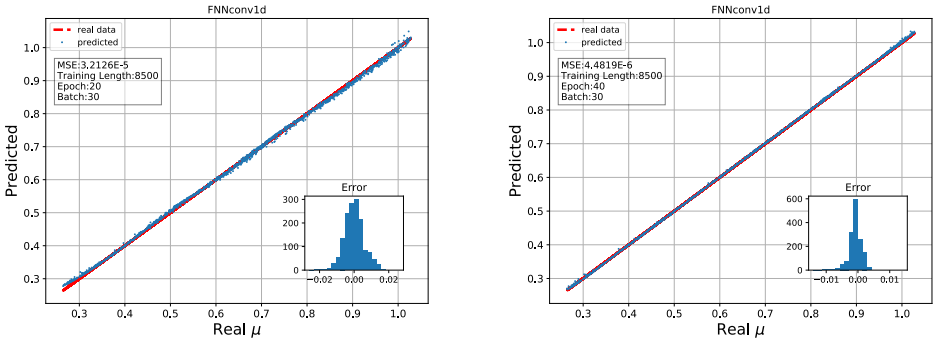


(c) FNN[128, 30, 30, 10, 4]



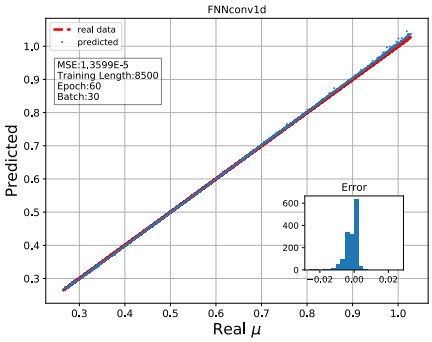
(d) FNN[128, 30, 30, 10, 4]

Figure 9: In first 20 epoch (a), there are relatively higher deviations in lower energies. In (b), deviations become smaller but still visible. In (c), prediction line is smoother but still deviations remains in lowest energy levels.

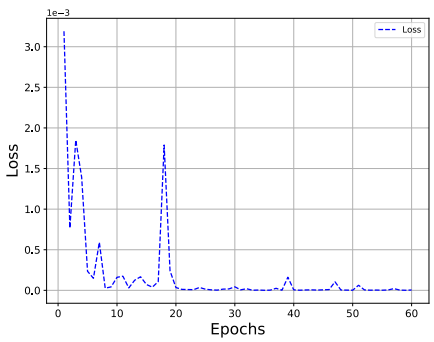


(a) Conv1D

(b) Conv1D

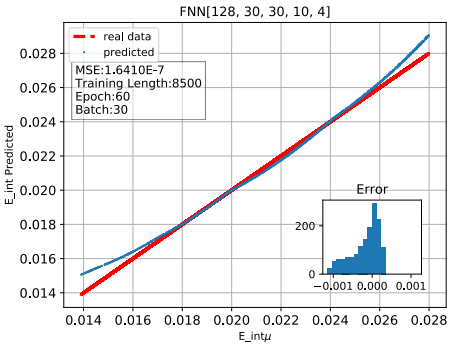


(c) Conv1D

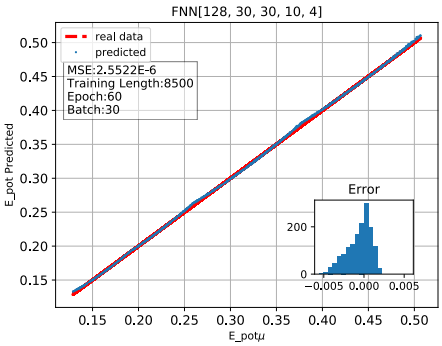


(d) Conv1D

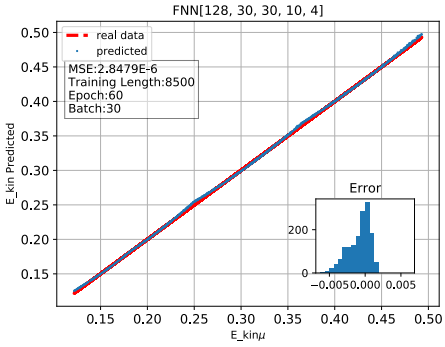
Figure 10: In (a) deviation has nearly same characteristic with FNN in lower energies. After 60 epochs (c), there is a visible shift in higher energies (after 0.7 (unit)) but this does not occur in (b). MSE also shows this fact since its order reduces.



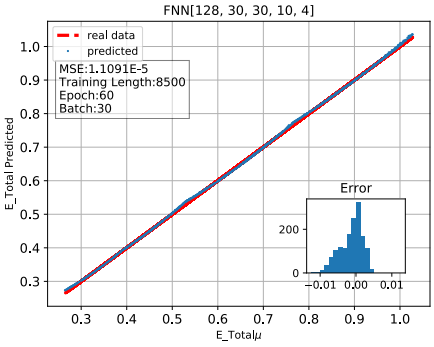
(a) FNN[128, 30, 30, 10, 4]



(b) FNN[128, 30, 30, 10, 4]



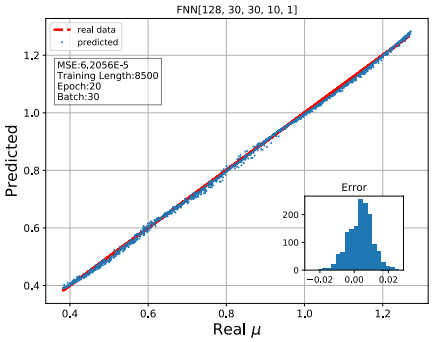
(c) FNN[128, 30, 30, 10, 4]



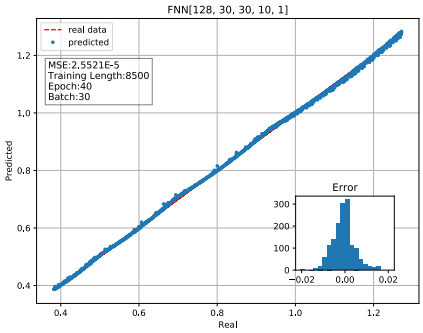
(d) FNN[128, 30, 30, 10, 4]

Figure 11: Notice that, the order of interaction energy is much smaller than the others. This may be the reason of bias in (a). Applying a proper normalization to the variables may reduce the error. **REF**

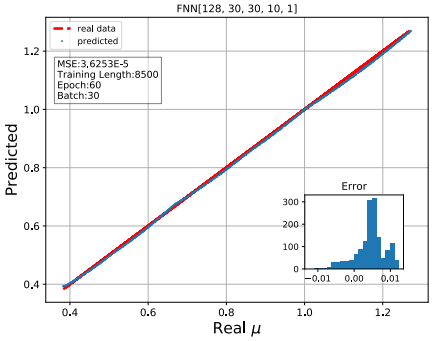
$\mathbf{G = 1}$



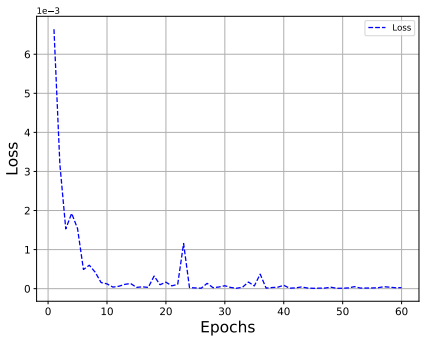
(a) FNN[128, 30, 30, 10, 4]



(b) FNN[128, 30, 30, 10, 4]

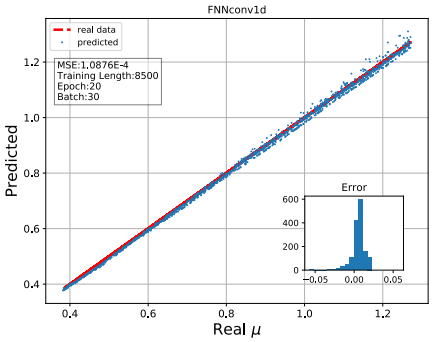


(c) FNN[128, 30, 30, 10, 4]

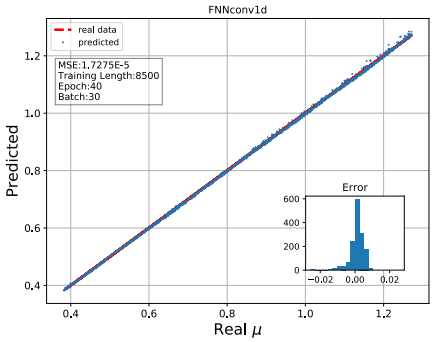


(d) FNN[128, 30, 30, 10, 4]

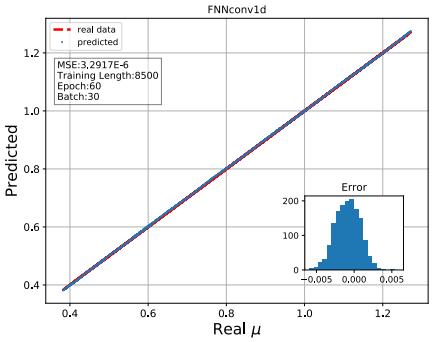
Figure 12: g = 1



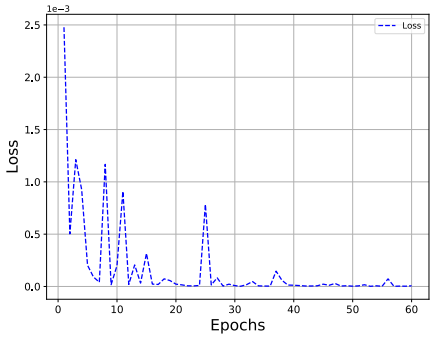
(a) Conv1D



(b) Conv1D

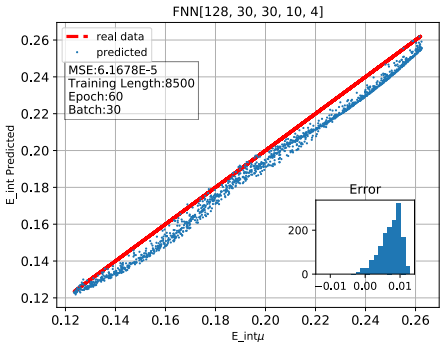


(c) Conv1D

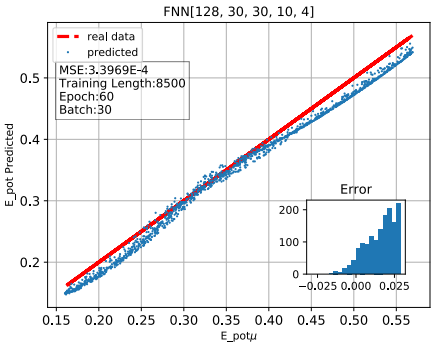


(d) Conv1D

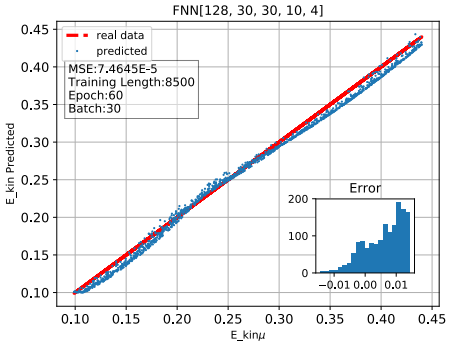
Figure 13: g = 1



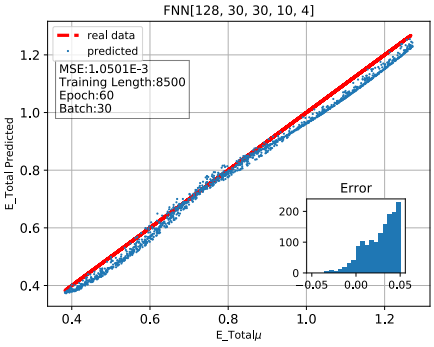
(a) FNN[128, 30, 30, 10, 4]



(b) FNN[128, 30, 30, 10, 4]



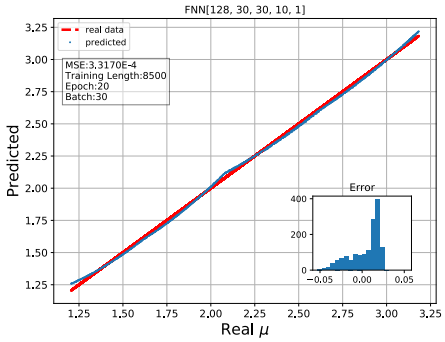
(c) FNN[128, 30, 30, 10, 4]



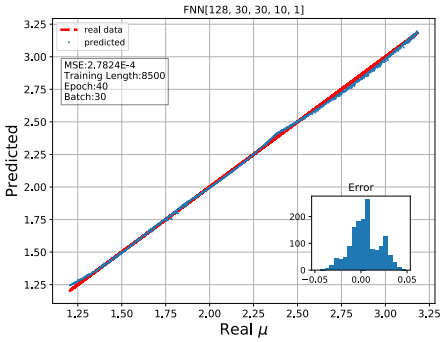
(d) FNN[128, 30, 30, 10, 4]

Figure 14: Separate prediction g=1

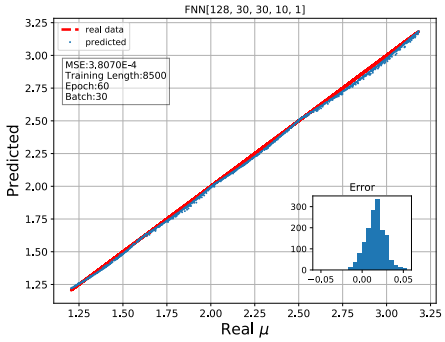
G=10



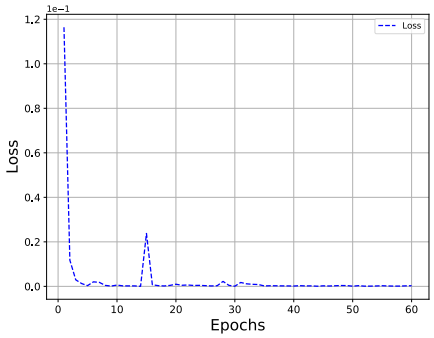
(a) FNN[128, 30, 30, 10, 4]



(b) FNN[128, 30, 30, 10, 4]

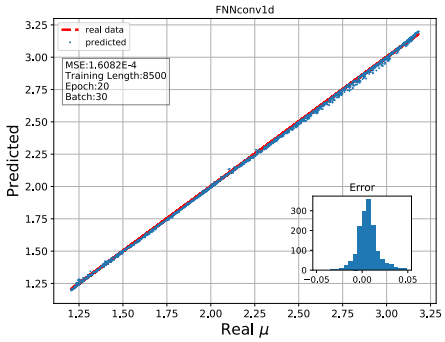


(c) FNN[128, 30, 30, 10, 4]

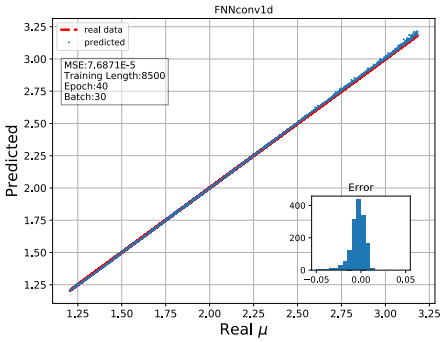


(d) FNN[128, 30, 30, 10, 4]

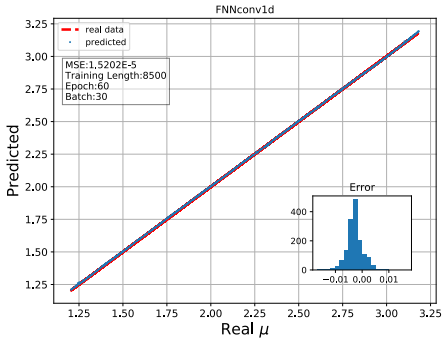
Figure 15: g = 10



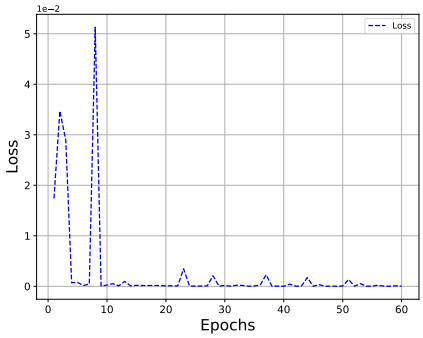
(a) Conv1D



(b) Conv1D

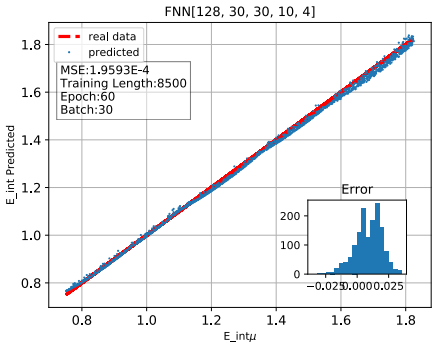


(c) Conv1D

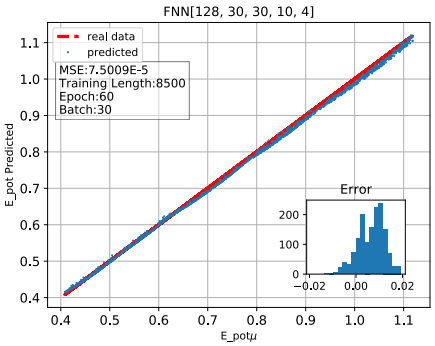


(d) Conv1D

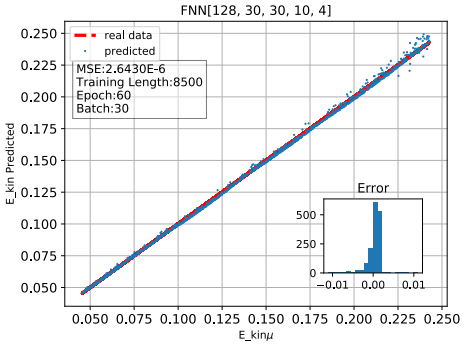
Figure 16: g = 10



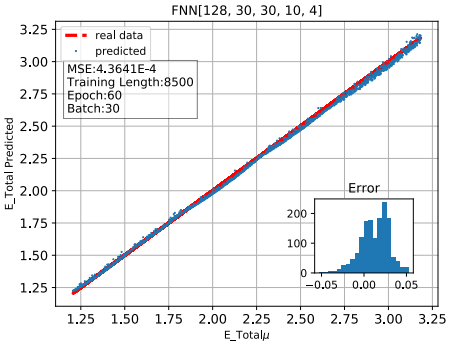
(a) FNN[128, 30, 30, 10, 4]



(b) FNN[128, 30, 30, 10, 4]



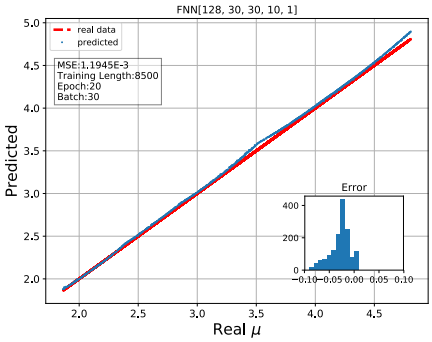
(c) FNN[128, 30, 30, 10, 4]



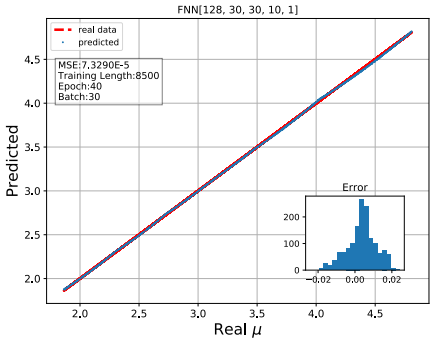
(d) FNN[128, 30, 30, 10, 4]

Figure 17: Separate prediction g=10

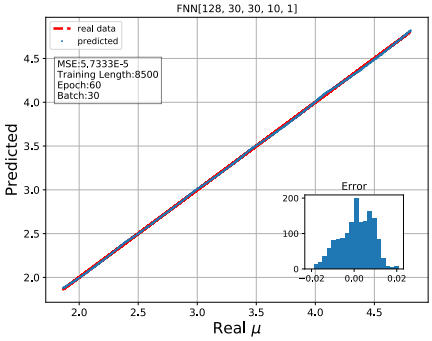
G = 20



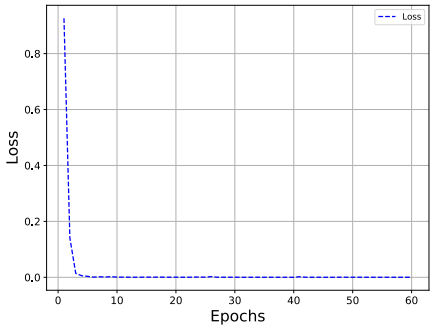
(a) FNN[128, 30, 30, 10, 4]



(b) FNN[128, 30, 30, 10, 4]

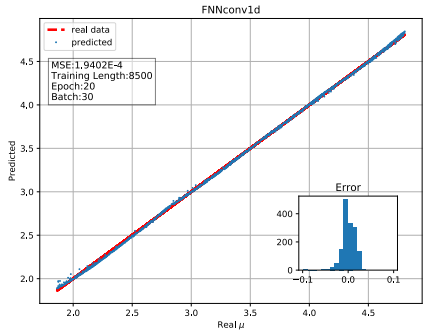


(c) FNN[128, 30, 30, 10, 4]

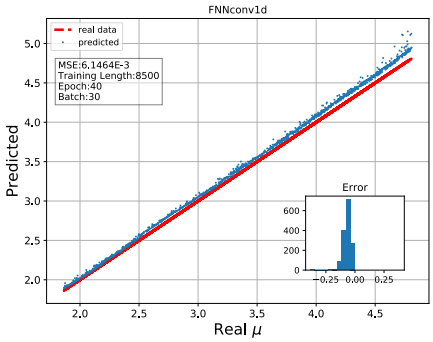


(d) FNN[128, 30, 30, 10, 4]

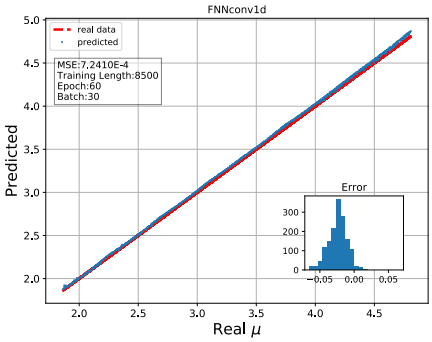
Figure 18: g = 20



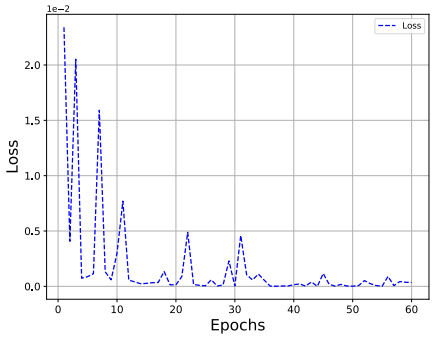
(a) Conv1D



(b) Conv1D

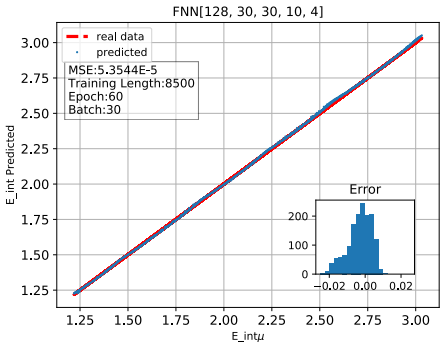


(c) Conv1D

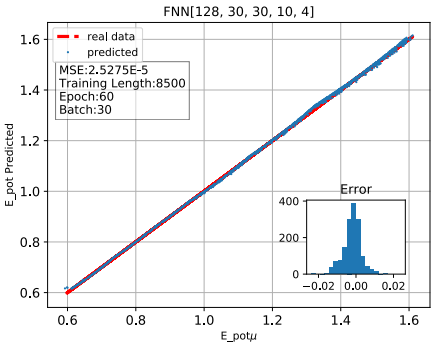


(d) Conv1D

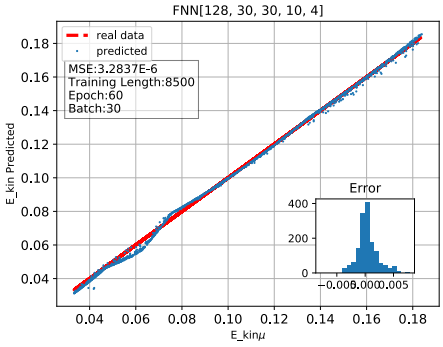
Figure 19: g = 20



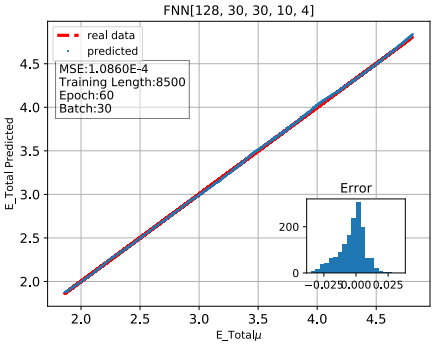
(a) FNN[128, 30, 30, 10, 4]



(b) FNN[128, 30, 30, 10, 4]



(c) FNN[128, 30, 30, 10, 4]



(d) FNN[128, 30, 30, 10, 4]

Figure 20: Separate prediction g=20

4.3.3 Inverse Problem Prediction of g

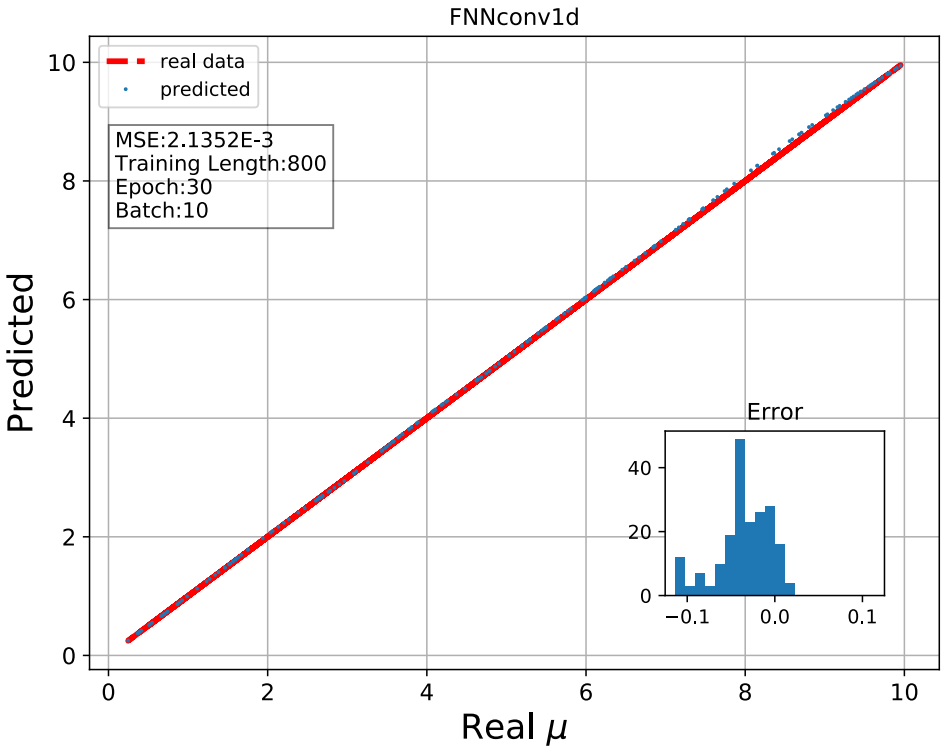


Figure 21: Fixed potential and shift

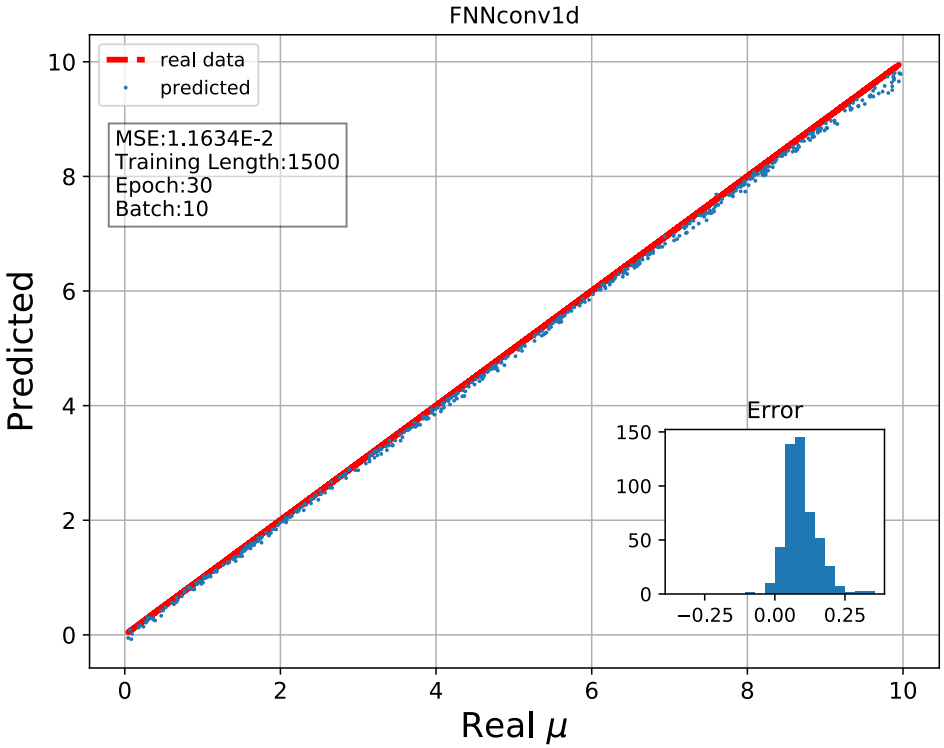


Figure 22: Various potential and fixed shift

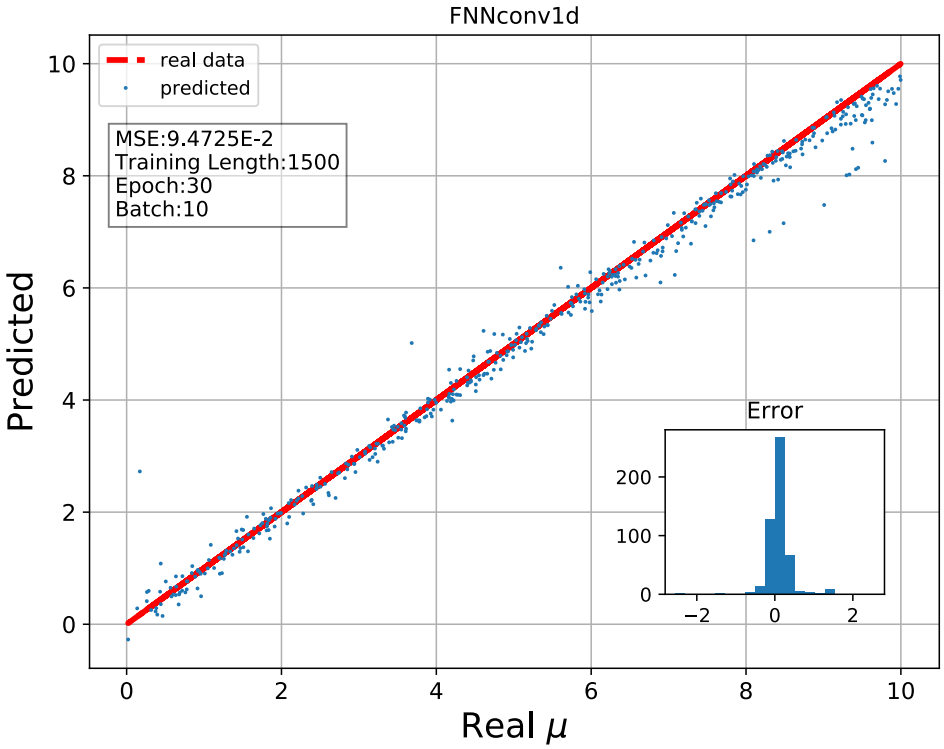


Figure 23: Various potential and shift