PHASE TRANSITION IN QUANTUM ISING MODEL

A PREPRINT

Yasamin Panahi
Department of Physics
Sharif University of Technology
Tehran, Iran
yasamin.s.panahi@gmail.com

Parsa Rangriz
Department of Physics
Sharif University of Technology
Tehran, Iran
rangriz@physics.sharif.edu

Asra Rezafadaei
Department of Physics
Sharif University of Technology
Tehran, Iran
gh.fadae1378@gmail.com

June 21, 2021

ABSTRACT

The transverse-field Ising models and their phase transitions are studied in the regimes of condensed matter physics and statistical physics. In this project, we sought to find the phase transition point in the one-dimensional transverse-field Ising model and classify different phases, using Machine Learning and Deep Learning methods. To do this, we first generated our data by solving the model numerically using the exact diagonalization technique, then computing the quantities that are helpful such as the correlation function, the entanglement entropy,etc. Then we applied some traditional techniques like KNN, Decision Tree, Kernel Ridge, and Support Vector Machine to find the best number for reporting as the phase transition point. Then we compared the results with analytical solution. In the end, we designed a neural network for our model and did similar procedures with different activation functions for this Quantum Ising model.

1 Introduction

Spin is a quantum mechanical property of an electron and is related to angular momentum. Since the 20th century, spin has became an important feature for physicists. One of the most important models which includes more and more spins is Ising model. One can calculate some variables such as energy, correlation functions, entanglement entropy, etc. But if we look closely to these models, we can find out a phase transition. Indeed, phase transition is a thermodynamic phenomenon and it occurs in thermodynamic limit of a system. Everyday we observe it in our life, for example melting and evaporating are just some simple examples. In other magnetic systems, the phase transition happens when the magnetic property of the model has changed, i.e, phase transition from paramagnetic order to ferromagnetic one.

Estimation is the process that we use to derive some conclusion from our data and known information. Consider a set of variables $\mathbf{x} = \{x_i\}_{i=1,\cdots,N}$ on which one is able to gain some partial observations $\mathbf{y} = \{y_\mu\}_{\mu=1,\cdots,M}$. Our aim is to infer the values of the variables $\mathbf{x}$ based on our observation. Since our model is a finite system (chain of N atoms) and we need the most possible exact numerical solution to obtain our data, we decided to use Exact diagonalization (ED) technique. It's a common numerical technique used in physics to determine the eigenstates and energy eigenvalues of a quantum Hamiltonian. In our problem, we have some data such as correlation function, energy differences, entanglement entropy, and we seek to find the phase transition point from them, by phase transition point we mean the h coefficient in hamiltonian that will determine the magnetic phase of system according to orientation of spins, Figure 1 shows these two magnetic phases for different ranges of h, if $h < 1$ it's Ferromagnetic and if $h > 1$ it's Paramagnetic.

2 Analytical Exact Solution

The Hamiltonian of the system is the following:

$$\mathcal{H} = -\sum_i (h\sigma_x^i + J\sigma_z^i\sigma_z^{i+1}) \tag{1}$$

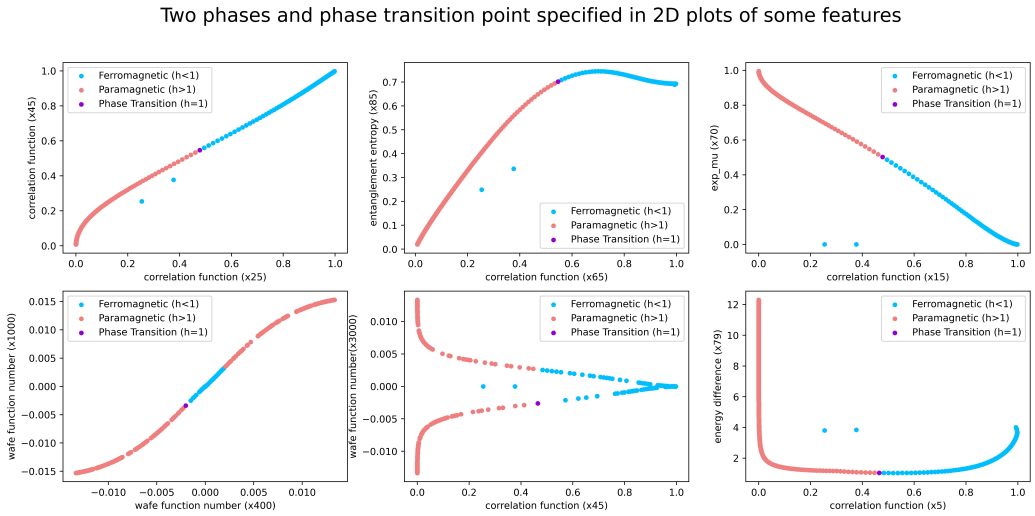


Figure 1: Two magnetic phases for different ranges of h.

where σ_αs are the Pauli spin operators. The spin operators satisfy the commutation relations given by (with $\hbar = 1$):

$$[\sigma_\alpha^i, \sigma_\beta^j] = 2i\delta_{ij}\epsilon_{\alpha\beta\gamma}\sigma_\gamma^k, \quad \alpha = x, y, z \tag{2}$$

and also they satisfy the anti commutation relations given by:

$$\{\sigma_\alpha^i, \sigma_\beta^i\} = \sigma_\alpha^i \sigma_\beta^i = 0 \tag{3}$$

and

$$(\sigma_\alpha^i)^2 = 1 \tag{4}$$

on the same site.

This transverse field Ising chain Hamiltonian can be exactly diagonalized, and the entire eigenvalue spectrum and eigenfunctions can be calculated by Jordan-Wigner transformation.

Using a canonical transformation:

$$\sigma_x \to \sigma_z, \quad \sigma_z \to -\sigma_x \tag{5}$$

then we have

$$\mathcal{H} = -\left(h\sum_i \sigma_z^i + J\sum_i \sigma_x^i \sigma_x^{i+1}\right) \tag{6}$$

Also, we can rewrite the Hamiltonian in terms of the raising and lowering operators σ_+^i and σ_-^i where

$$\sigma_+^i = \frac{1}{2}(\sigma_x^i + i\sigma_y^i) \tag{7}$$

$$\sigma_-^i = \frac{1}{2}(\sigma_x^i - i\sigma_y^i) \tag{8}$$

By using Jordan-Wigner transformation, we define

$$c_i = \prod_{j=1}^{i-1} \exp\left(i\pi\sigma_+^j\sigma_-^j\right)\sigma_-^i \tag{9}$$

$$c_i^\dagger = \sigma_+^i \prod_{j=1}^{i-1} \exp\left(-i\pi\sigma_-^j\sigma_+\right) \tag{10}$$

These creation and annihilation operators, c_i and $c_i^\dagger$ satisfy:

$$\{c_i, c_i^\dagger\} = \delta_{ij} \tag{11}$$

$$\{c_i, c_j\} = \{c_i^\dagger, c_j^\dagger\} = 0 \tag{12}$$

The Pauli spin operators, σ_x^i and σ_z^i are then transformed as follows:

$$\sigma_x^i = 1 - 2c_i^\dagger c_i \tag{13}$$

$$\sigma_z^i = -\prod_{j=1}^{i-1}(1 - 2c_j^\dagger c_j)(c_i^\dagger + c_i) \tag{14}$$

Thus the Hamiltonian becomes:

$$\mathcal{H} = -J\sum_{i=1}^{N}(c^i c_{i+1} + \sigma + \sigma_{i+1}^\dagger c_i + \sigma_i^\dagger c_{i+1}^\dagger + c_{i+1}c_i + \frac{h}{J}(1 - 2c_i^\dagger c_i)) \tag{15}$$

The above equation is already quadratic in the fermion operators and it is obviously diagonalizable. Therefore, let us consider fermions in momentum space (Fourier space):

$$c_k = \frac{1}{\sqrt{N}}\sum_{j=1}^{N} c_j \exp(ikR_j) \tag{16}$$

$$c_k^\dagger = \frac{1}{\sqrt{N}}\sum_{j=1}^{N} c_j^\dagger \exp(-ikR_j) \tag{17}$$

where $k = \frac{2\pi m}{N}$ and $m = \frac{-N-1}{2}, \cdots, 0, \cdots, \frac{N-1}{2}$ for even N and $m = \frac{-N}{2}, \cdots, 0, \cdots, \frac{N}{2}$ for odd N. The final Hamiltonian in momentum space is:

$$\mathcal{H} = J\sum_{k}(2(\frac{h}{J} - \cos(k)c_k^\dagger c_k - i\sin(k)(c_{-k}^\dagger c_k + c_{-k}c_k)) \tag{18}$$

to diagonalize this Hamiltonian, we should use the Bogoliubov transformation,

$$\gamma_k = u_k c_k - iv_k c_{-k}^\dagger \tag{19}$$

where γ_k is a new fermionic operator and u_k and v_k are real numbers which satisfy

$$u_{-k} = u_k, \quad v_{-k} = v_k \tag{20}$$

$$u_k^2 + v_k^2 = 1 \tag{21}$$

By this transformation, finally the Hamiltonian can be diagonalized:

$$\mathcal{H} = \sum_k \epsilon_k(\gamma_k^\dagger\gamma_k - \frac{1}{2}) + const. \tag{22}$$

$$\epsilon_k = 2J(1 + \frac{h^2}{J^2} - 2\frac{h}{J}\cos(k))^{\frac{1}{2}} \tag{23}$$

where the constant is the zero-point energy of the spinless fermion system, which is given by

$$E_0 = -\sum_k \epsilon_k \tag{24}$$

If we plot the elementary excitations as a function of wave vector k for different values of h (Figure 1), we can see there is an energy gap in the excitation spectrum of the system at $k = 0$ for $h_c = 1$.

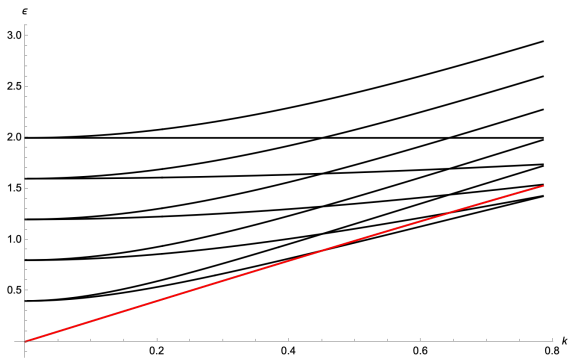


Figure 2: Elementary excitation energy as a function of k for different h. The red one is $h = 1$ and the other ones are the other h between 0 until 2

3 Data Generation

3.1 Exact diagonalization technique

Since our model is a finite system (chain of N atoms) and we need the most possible exact numerical solution to obtain our data, we decided to use Exact diagonalization (ED) technique. It's a common numerical technique used in physics to determine the eigenstates and energy eigenvalues of a quantum Hamiltonian. In this technique we should express the Hamiltonian in a matrix form in order to diagonalize it by computer. First we need to construct pauli matrices to build the Hamiltonian, in this part we used sparse structure to minimize memory cost and speed up the code. Again, we write the Hamiltonian of the system which is

$$\mathcal{H} = -\sum_{i=1}^{N}(h\sigma_x^i + J\sigma_z^i\sigma_z^{i+1}) \tag{25}$$

where we choose $N = 12$, $J = 1$, and $h \in [0, 7]$. We want to find the phase transition point, called h_c and we expect it to be $h = 1$ as the analytical solution predicts. In our Hamiltonian, we have tensor products of Pauli matrices to find the spin operators for each site so

$$\sigma_z^i = \underbrace{I \otimes I \otimes \cdots \otimes I}_{\text{i-1}} \otimes \sigma_z \otimes \underbrace{I \otimes I \otimes \cdots \otimes I}_{\text{N-i}} \tag{26}$$

$$\sigma_x^i = \underbrace{I \otimes I \otimes \cdots \otimes I}_{\text{i-1}} \otimes \sigma_x \otimes \underbrace{I \otimes I \otimes \cdots \otimes I}_{\text{N-i}} \tag{27}$$

The next step is to diagonalize Hamiltonian by using *eig* function in MATLAB. This is the most important part of our numerical solution because we'll need the ground state wave function in order to calculate almost all of our features, as we know diagonalization means

$$\mathcal{H}\,|\psi_n\rangle = E_n\,|\psi_n\rangle \tag{28}$$

where $|\psi_n\rangle$ is the nth eigenvector (wave function) and E_n is the $n$the eigenvalue (energy). We seek to find the ground state. So we should find the minimum of E_n and call it E_{min} and the ground state $|\psi_{min}\rangle$ = $|g\rangle$. We should mention that we have imposed periodic boundary conditions to our system.

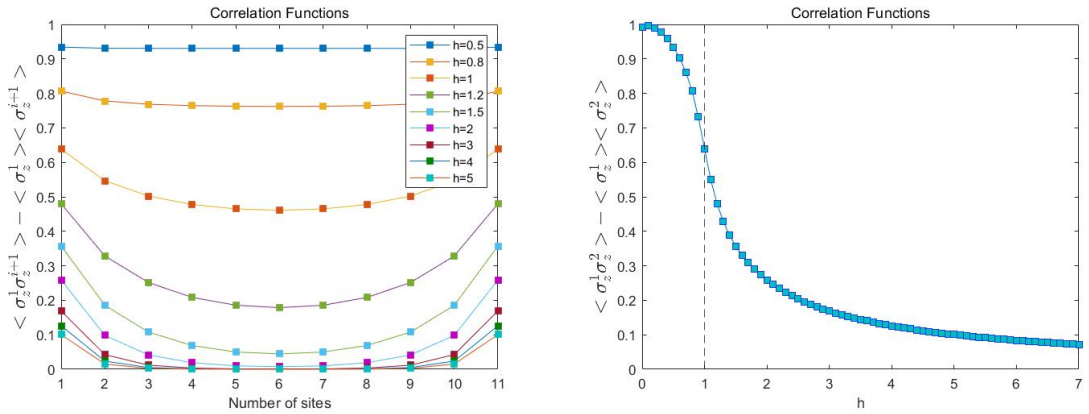
3.2 Data

3.2.1 How the data is collected and what they physically mean?

Now that we have the ground state wave function, we begin to calculate physical quantities that will help us see the phase transition in our model. These quantities will be different features of our data. The most important feature is the spin - spin correlation functions (Figure 2):

$$G(\sigma_z^i, \sigma_z^{i+1}) = \left\langle \sigma_z^i \sigma_z^{i+1} \right\rangle - \left\langle \sigma_z^i \right\rangle \left\langle \sigma_z^{i+1} \right\rangle \tag{29}$$

where $\langle C\rangle = \langle g|\,C\,|g\rangle$. As we can see in Figure 2.a , the visible gap between the $h = 1$ plot and other plots is a sign of phase transition in our model, also in Figure 2.b we can see that the $h = 1$ dashed line is like a vertical asymptote which is another sign to know that correlation function is a good feature. The next useful quantity is the mean of σ_x (Figure

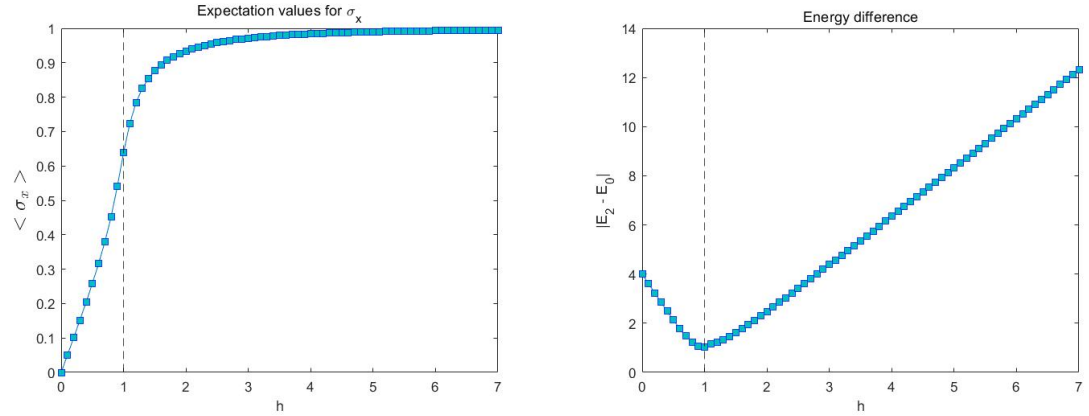


(a) Correlation functions of the first site as a function of its distance with others.

(b) Correlation function between the first and second site as a function of h

Figure 3: spin - spin Correlation functions

3.a). In the other word, we try to find $\langle \sigma_x^i \rangle$. As we can see in Figure 3.a , again the $h = 1$ dashed line is like a vertical asymptote which is a sign for phase transition. The other helpful feature that we can calculate just by diagonalizing the Hamiltonian is the difference between the second excited state energy and the ground state energy (Figure 3.b). As it's shown in this Figure, this energy difference has it's minimum value at $h = 1$ which is the phase transition point.



(a) $\langle \sigma_x \rangle$ as a function of h.

(b) E_2 - E_0 as a function of h.

Figure 4: Expectation value of σ_x and the Energy difference

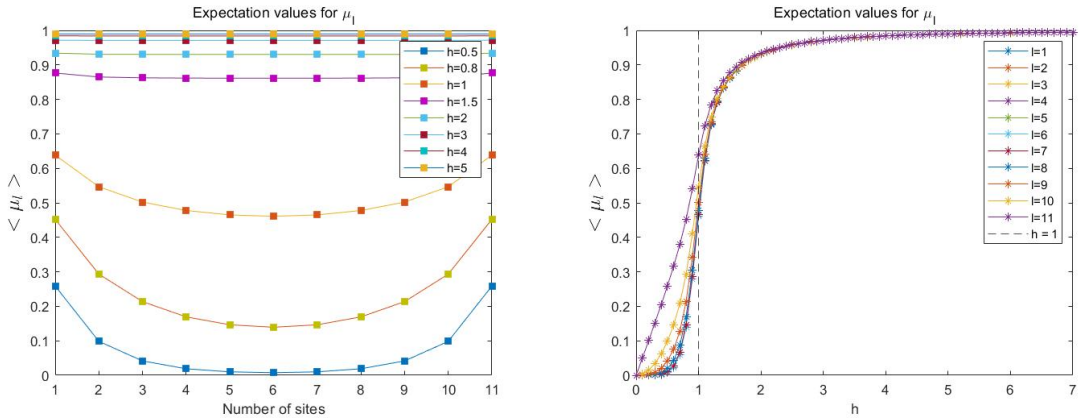
Considering that we have a transverse field applied to the system we guess that another good feature can be calculated by using σ_x . we'll call it μ_l (Figure 4) and again we calculate its expectation value which is defined by

$$\langle \mu_l \rangle = \left\langle \sigma_x^1 \sigma_x^2 \cdots \sigma_x^l \mathbb{I}^{l+1} \cdots \mathbb{I}^N \right\rangle \tag{30}$$

Similar to the spin - spin correlation functions Figures, there's a gap between $h = 1$ plot and other plots in Figure 4.a and also in Figure 4.b the $h = 1$ dashed line is like a vertical asymptote which another sign for phase transition.

Other useful feature is Entanglement entropy for different subsystems in our chain (Figure 5). The first step is to find the Reduced Density Matrix (RDM) for $N - 1$ possible subsystems. One easy method for this calculation is to reshape the ground state wave function in a form that is suitable for each subsystem. For example if we want to divide our chain to N_{left} spins and N_{right} spins ($N = N_{left} + N_{right}$) and then want to calculate reduced density matrix for N_{left} spins, our Hilbert space is $2^{N_{left}} \times 2^{N_{left}}$. So we reshape our $2^N \times 1$ ground state wave function to a $2^{N_{right}} \times 2^{N_{left}}$ matrix and call it $\tilde{\psi}_{left}$, hence we can calculate the reduced density matrix (ρ) by writing

$$\rho_{left} = \tilde{\psi}_{left}^{\dagger} \tilde{\psi}_{left} \tag{31}$$



(a) $\langle \mu_l \rangle$ as a function of number of sites

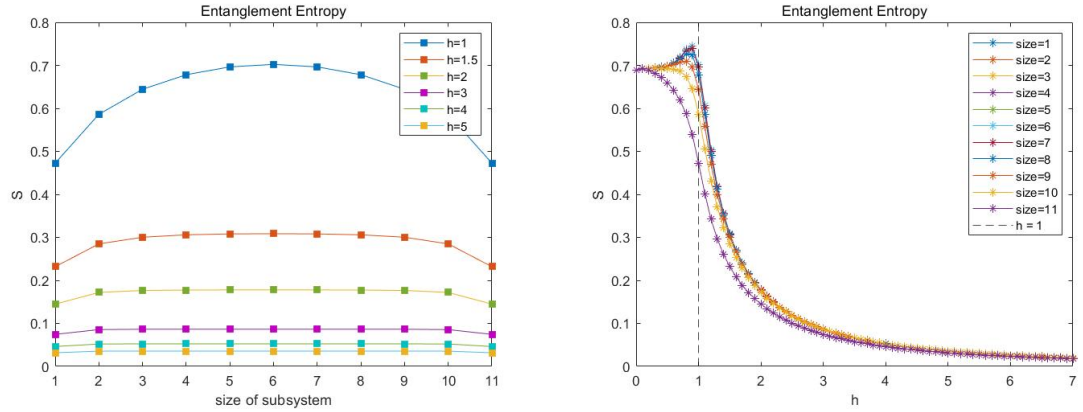
(b) $\langle \mu_l \rangle$ as a function of h

Figure 5: Expectation value of μ_l

and we can easily see that it is a $2^{N_{left}} \times 2^{N_{left}}$ matrix. Then we diagonalize this ρ_{left} in order to find its eigenvalues (ω) and now we are able to calculate the Entanglement entropy using

$$S = -\sum_k \omega_{k,left} \log \omega_{k,left} \tag{32}$$

we repeat this procedure for all $N-1$ subsystems.



(a) Entanglement entropy as a function of size of subsystem.

(b) Entanglement entropy as a function of h.

Figure 6: Entanglement entropy

Apart from the gap for $h = 1$ plot in Figure 5.a, in Figure 5.b we see that the Entanglement entropy values drop at the phase transition point ($h = 1$) as it's shown with the dashed line, this is an evidence that the Entanglement entropy can also be cosidered a good feature. Now it seems that we have calculated many helpful features, but we know that the ultimate information about our system can be obtained from the ground state wave function and we haven't used it entirely unless we calculate all of the Higher order correlation functions and all other possible quantities, since it's not an efficient option due to the Computational cost that significantly increases the runtime, we decided to keep the ground state wavefunction (its 2^N numbers) so that we won't lose any substantial information about our model.

3.2.2 Cleaning up the data

As we mentioned before, we chose $h \in [0, 7]$. we covered this interval with small steps (0.01) and we calculated our features for different h's. So there're 701 samples in our data. Also we have 4186 features including spin - spin

Correlation function, Expectation values for σ_x and μ_l, the Energy difference ($E_2 - E_0$), Entanglement entropy (S) and Wavefunction numbers. These are different labels in our data. Finally we cleaned up our data in form of a 701×4187 matrix. There are respectively 66 Correlation fuctions, one $\langle \sigma_x \rangle$, 11 $\langle \mu_l \rangle$, one Energy difference, 11 Entanglement entropy and 4096 ground state Wave function numbers. The last coloumn is y which is h in our model.

4 Traditional Techniques

There are several models that we can use to learn the function for our problem. But in this project, we use just 5 of them which we assumed are more efficient and another one (Linear Regresion) for sake of curiosity!. We use the sci-kit learn library in Python to estimate and with help of its tutorial we introduce a very simple explanation of these models. Since our problem is a regression problem (finding h in hamiltonian) all the six models that we used are regression models.

4.1 Models

4.1.1 K-Nearest Neighbours (KNN)

The k-nearest neighbours algorithm is one of the most popular methods that is used for classification and regression. In the classification method, the points are classified by assigning the label which label is most frequent among the k training samples nearest to that query point. But in the regression mode, this algorithm is used for estimating continuous variables. First we compute the distance from the query example to the labeled examples. Then we order the labeled examples by increasing distance. At the end we find a optimal number of k of nearest neighbors, based on root mean square score error and calculate an inverse distance weighted average.

4.1.2 Decision Tree

Decision tree make the models in the form of a tree. It breaks a data into smaller subsets and we call the final result as decision nodes and leaf nodes. In this algorithm, we try to sketch a curve with a set of it-then-else decision rules. There are several methods such as ID3, C5.0 and CART for learning. One of them which it can be used for regression is CART. It constructs binary trees using the feature and threshold that yield the largest information gain at each node.

4.1.3 Gaussian Process Regression (GPR)

Gaussian process regression is a non-parametric, Bayesian approach to regression. Unlike many popular supervised algorithms that learn exact values for every parameter in a function, the Bayesian approach infers a probability distribution over all possible values. Indeed GPR calculate the probability distribution over all admissible functions that fit the data.

4.1.4 Kernel Ridge

Ridge regression is one of the linear models in which the target value is expected to be a linear combination of the features. It addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients.

Kernel ridge regression is a non-parametric form of ridge regression. The goal is to estimate a function by minimizing a squared loss with a squared norm regularization term.

4.1.5 Support Vector Machine (SVM)

Support vector machine is a supervised learning model that is used for classification and regression. In this algorithm, the aim is to find the best curve to have a maximum distance between the points of the data and the curve that we called it decision boundary. Then it classifies the data respect to this curve. The problem of regression is to find a curve that can predict the behaviour of the data. Consider these two red lines as the decision boundary and the green line as the hyper-plane. Our goal is to have the maximum number of the points that are within the decision boundary and the best fit line is the hyperplane that has a maximum number of points.

4.1.6 Linear Regression

Linear Regression, as we can see in its name, is a linear model to estimate from the actual data. The goal is to find coefficients $w = (w_1, \cdots, w_p)$ to minimize the residual sum of squares between the observed target in the data and the targets predicted by the linear approximation.

4.2 Metrics

The first step that we have to do is to set a metric. There are four common metrics in regression models. That we describe them in the following:

4.2.1 Mean Square Error (MSE)

If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the true value, then the mean squared error function is defined as follows:

$$MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2 \tag{33}$$

4.2.2 Mean Absolute Error (MAE)

If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the true value, the mean absolute error (MAE) estimated over N samples is defined as

$$MAE(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N-1} |y_i - \hat{y}_i| \tag{34}$$

4.2.3 R^2 score

If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the true value, the estimated R^2 is defined as:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y}_i)^2} \tag{35}$$

where $\bar{y} = \frac{1}{n} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$

4.2.4 Root Mean Square Error (RMSE)

The definition of this metric is similar to MSE, but there is a bit difference:

$$RMSE(y, \hat{y}) = \sqrt{MSE(y, \hat{y})} \tag{36}$$

We used all these four regression metrics and the results were quite compatible, for example when MSE, RMSE and MAE metrics which are error were too small, the R^2 metric which is score was quite high, There is no zero number or any other problamatic issue in our data set, so all these metrics were fine. But we had some preference to consider, First although the R^2 metric is better used to explain the model to other people since it does not take into consideration of over-fitting problem we refused to use it in the next section. the next point is that the MSE gives larger penalisation to big prediction error by square it while MAE treats all errors the same, so we prefer MSE to MAE. Also the values of our data is not too big to use RMSE so we prefer to use MSE as our main metric and the loss function for our validation curves and learning curves.

4.3 Performance of Model

4.3.1 Gaussian Process Regressor: Best Model

For the first fitting we fixed alpha = 1 and the MSE metric was about 0.01, we also plot GPR predictions for some of the features and compared it with actual data (Figure 9), it's not a very good prediction with this alpha but we can make it better when we fine tune the model.

4.4 Tuning the Models

In the third step, in order to find the best model, we use two techniques: Grid Search and Validation. Grid Search is a method in which by calculating the probable values of the estimator, we can get a table of all permutation of different values and their scores, evaluated by a MSE metric. Validation, is a way that ensures the parameters we have chosen, do not end up with over-fitting or under-fitting.

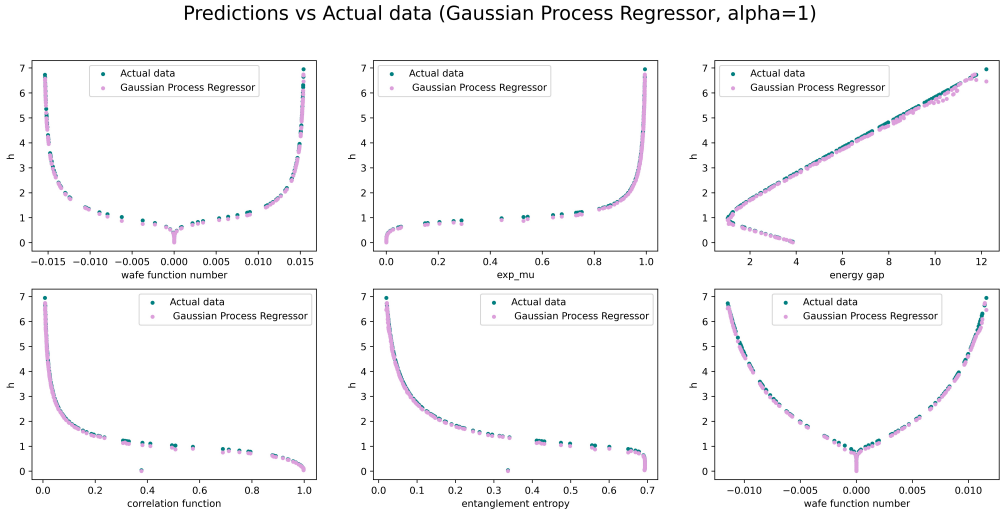


Figure 7: Actual data vs GPR predictions

4.5 Validation Curve

These are validation curves for different models that we used.we can see which from these curves that which values are optimal for complexity of our models. (see Figure 11)

4.6 Learning Curve

After finding the best hyper-parameters of our model, we plot the learning curve which is loss vs sample size, to see if we have enough data or not. The following plots are learning curves for our 6 models using MSE as loss function, there are also two other curves, one shows fit time vs sample size and the other is MSE vs fit time, the third curve is very useful when we want to compare performance of models. (see Figures 12 - 17)

4.6.1 Bias and Variance Estimation

Whenever we want to comparison model prediction, it's very important to understand prediction errors (bias and variance). There is a trade-off between a model's ability to minimize bias and variance. Now let's start with the definition of these ideas and see how they make difference. Bias is the difference between the average prediction of our model and the correct value. Model with high bias pays very little attention to the training data. It always leads to high error on training and test data. Variance is the variability of model prediction for a given data point or a value which tells us spread of our model. In fact, model with high variance pays a lot of attention to training data and doesn't generalize on the data which it hasn't seen before. (see Figure 20)

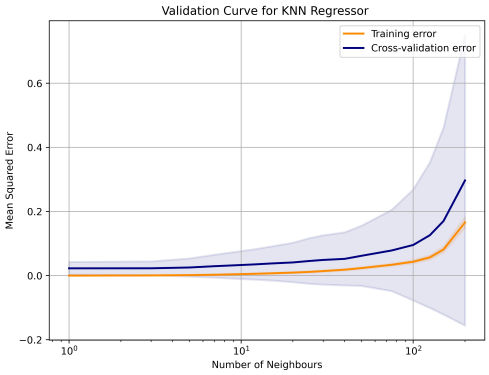
Now we have to estimate the bias and variance for our models.

Model	Bias	Variance
KNN Regressor	e-04	0.006
Decision Tree Regressor	e-04	0.007
Gaussian Process Regressor	e-06	0.006
Kernel Ridge Regressor	e-05	0.005
Support Vector Regressor	0.005	0.005

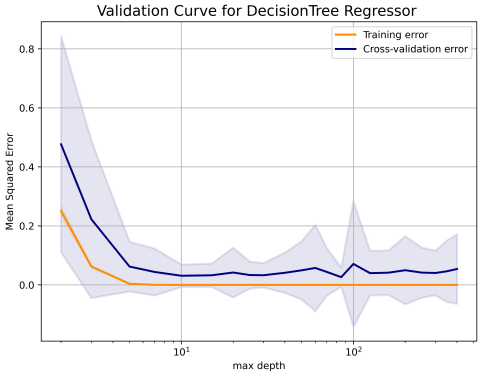
5 Feedforward Neural Network

5.1 What is FNN and why we used this model?

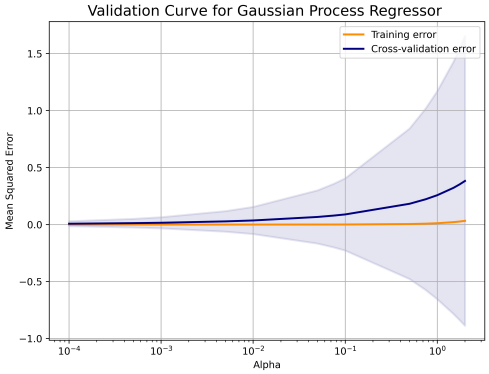
A Feed Forward Neural Network is an artificial neural network in which the connections between nodes does not form a cycle. The feed forward model (Figure 2) is the simplest form of neural network as information is only processed in one



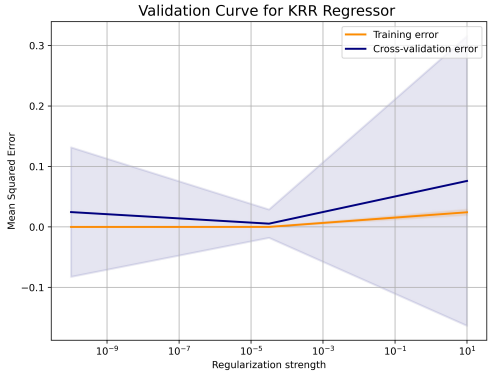
(a) KNN Regressor Validation curve



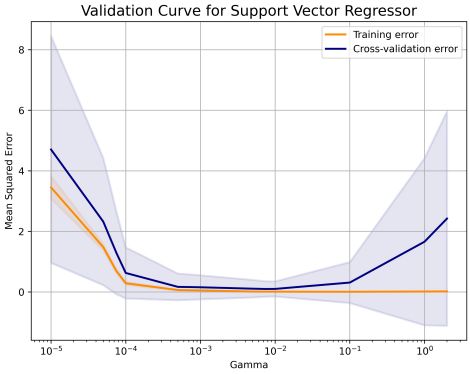
(b) Decision Tree Regressor Validation curve



(c) Gaussian Process Regressor Validation curve



(d) Kernel Ridge Regressor Validation curve



(e) Support Vector Regressor Validation curve

Figure 8: Validation Curves

direction. While the data may pass through multiple hidden nodes, it always moves in one direction and never backwards. In this model, a series of inputs enter the layer and are multiplied by the weights. Each value is then added together to get a sum of the weighted input values. Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN) are the others offered and useful neural network models, RNN is usually used to analyze time-dependent data set but since our data set is time-independent, RNN isn't useful for our problem. Also CNN commonly is used to analyze visual imagery data set and again this model isn't useful for our problem.In this Report we show that a standard feed-forward neural network can be trained to detect two phase of matter and find the critical point of Quantum ising model perfectly. An important part of neural network is their activation function. The neural network activation functions, in general,

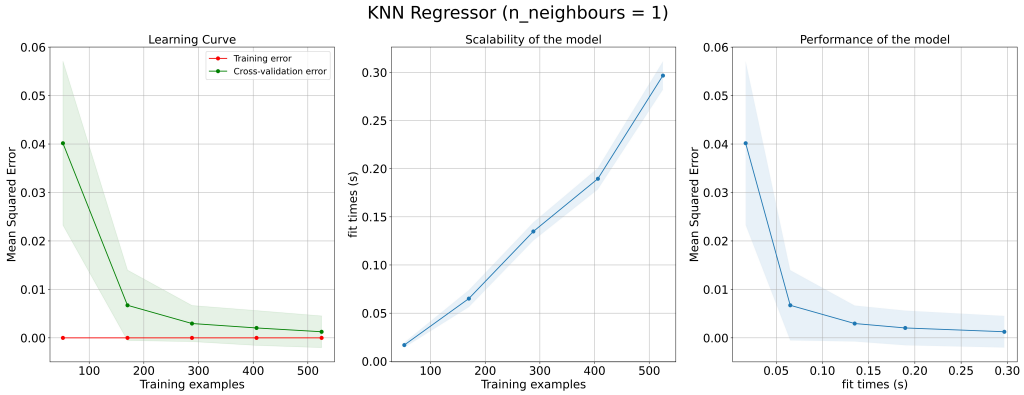


Figure 9: KNN Regressor Learning Curves

Figure 10: Decision Tree Regressor Learning Curves

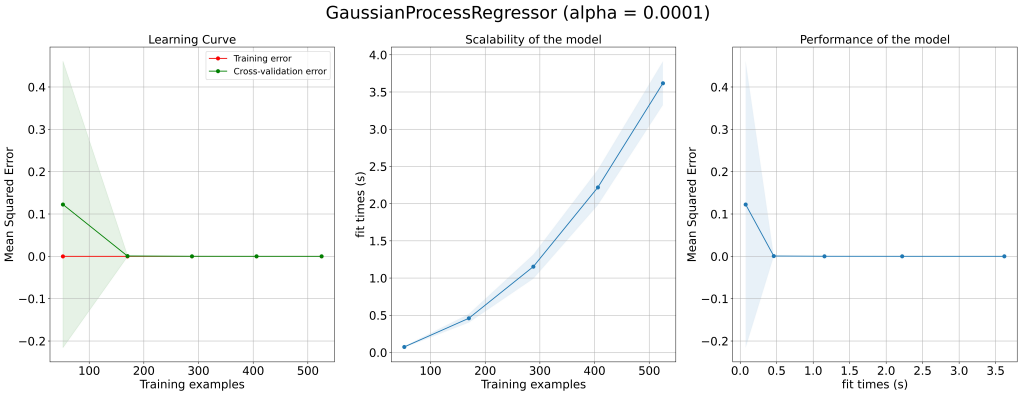


Figure 11: Gaussian Process Regressor Learning Curves

are the most significant component of Machine Learning, they are fundamentally used for determining the output of deep learning models, its accuracy, and performance efficiency of the training model that can design or divide a huge scale neural network. In this project we used tanh, softmax, softsign, relu and sigmoid as activation function. For implementation of the neural network in python we used keras module from tensorflow library to make a sequential model where each layer has exactly one input tensor and one output tensor.

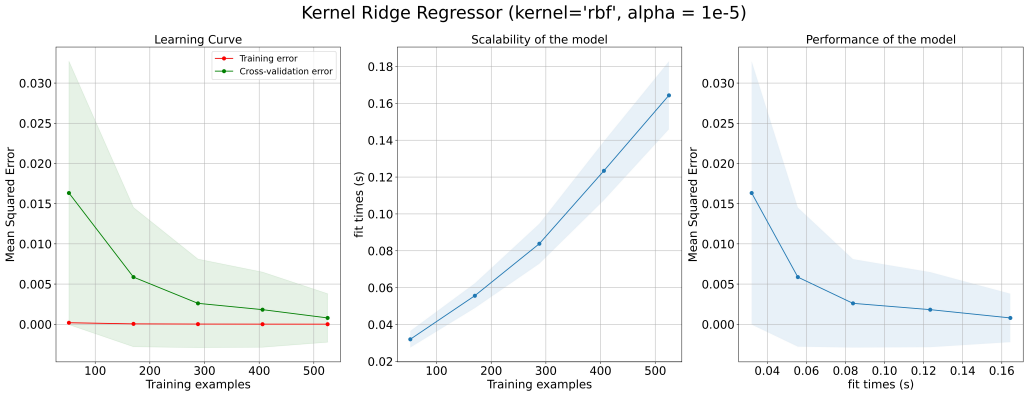


Figure 12: Kernel Ridge Regressor Learning Curves

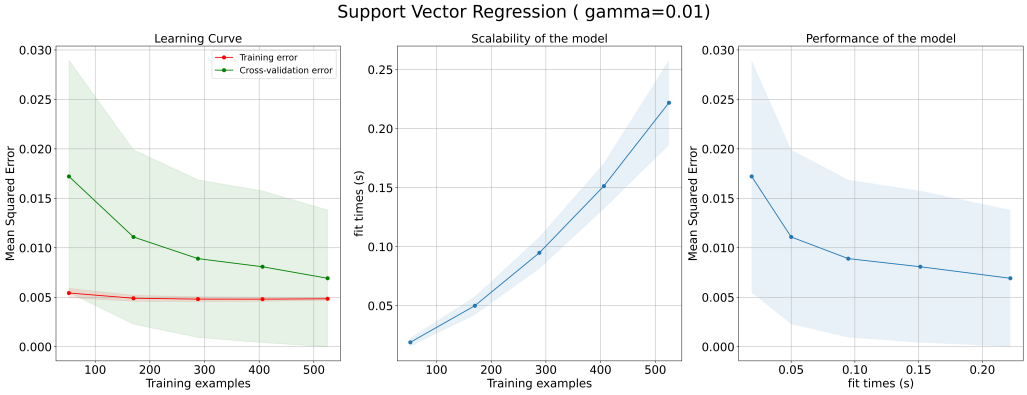


Figure 13: Support Vector Regressor Learning Curves

6 Evaluation

For using neural network, we have to set some layers and activation functions. But this won't be possible unless we evaluate the performance of each neural network we have used. In the following section, we will explain how we did this process for both Regression and Classification Problems. This Part includes the Loss vs Epoch curves for regression and Loss & Accuracy vs Epoch curves for classification the proper metrics for each problem are also discussed. In the first part, our goal was only to design a Neural Network to predict our desired results well, so we didn't tune the model,

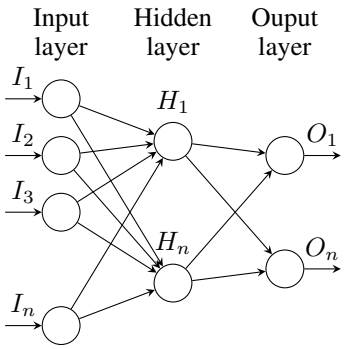


Figure 14: A Simple Feedforward Neural Network

we just played with different hyperparameters, altough we didn't use tuning in this section the results of our designed Neural Network's were quite satisfactory.

6.1 Regression

The main purpose of our problem was to find h which determines phase of the system. Our regression problem is not a regression for vlaues between 0 and 1 as for this case the sigmoid function for the last layer of Neural Network would be a good choice but since the 'y' label in our data set can be a decimal value between 0 and 7, our probelm is regression to arbitrary values, so the proper choice for the last layer of the Feedforward Neural Network would be a linear function with single node (since it's regression). We trained different Feedforward Neural Networks for different options and situations. First we traind three FNN's with same activation function for all hidden layers, One FNN which all hidden layers are relu, one with tanh and another one with sigmoid. Next we designed a FNN with differnt activation functions containing Relu, Tanh, Softsign and Softmax. We tried optimizers like SGD, Adam and RMSprop. After trying all these three options we concluded that the Adam optimizer works significantly better than others for our problem, so when compiling the model with keras, we set optimizer to be Adam. In order to see the predictions of our Neural Networks for 'h', we ploted 'h' vs some of our features which are shown in Figures 4, 5, 6. As we see in the figures the prediction vs. actual data curves are nearly fitted and neural networks work well.

6.1.1 Metrics and Loss vs Epoch Curve

As we mentioned in the privious section, we chose MSE as the main metric for our problem so for compiling the model we set loss to be mean squared error. After triaing and testing the model we used all four main regression metrics expressed in privious report (MSE, MAE, RMSE and R^2 score) to evaluate the prediction of our designed Neural Networks. We also plot the Loss vs Epoch curve for our train and test data using MSE as loss fucntion (Figure 3). This curve is quite informative, by looking at this curve we can see how the model is working on our test and train data and whether the model overfits or not, Fortunately we did not come across any over-fitting or under-fitting.

6.2 Classification

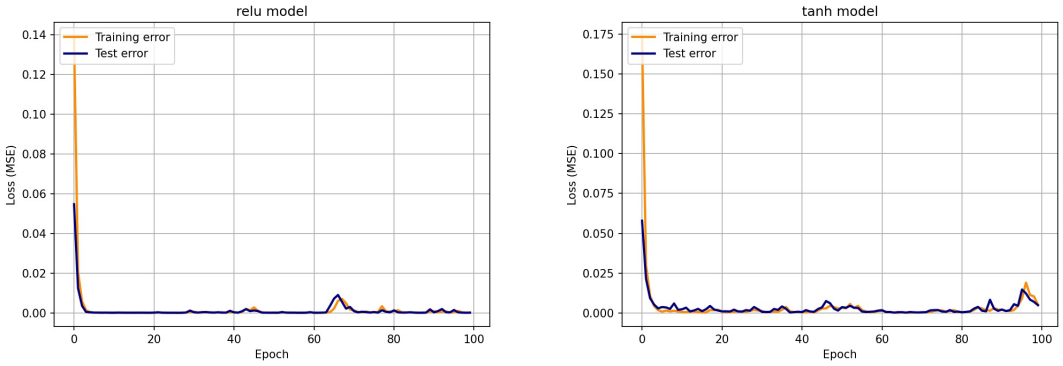
The main goal in our project was to find the 'y' lable of our data set to determine the phases of system, we treated this goal as a regressrion problem in the previous milestone.But now for the third milestone with Neural Network we decided to solve both regression and classification problems. In order to solve our problem from classification point of view we delete the phase transition point in our data set and assumed that we have only samples with $h < 1$ and $h > 1$ labels which we want to classify. For this purpose we assign 0 to all $h < 1$ samples (Ferromagnetic Phase) and 1 to all $h > 1$ samples (Paramagnetic Phase), so it's a binary classification problem. We designed a FNN with different activation functions containing Relu, Tanh, Softsign and Softmax, the optimizer was Adam and the loss function and metric were respectively binary cross entropy and accuracy. In order to see how the model worked for classification we ploted the classification results for different features which can be seen in the notebook of our code.

6.2.1 Loss & Accuracy vs Epoch Curve and Confusion Matrix

Like the regression part we also plot the Loss vs Epoch curve for out train and test data, this time using binary cross entropy loss fucntion, since we have classification we also plot the Accuracy vs Epoch plot (Figure 14) one of these plots is before tuning and the other two are after tuning, as we can see in these plots there isn't any over-fitting or under-fitting here. One of the best and most informative classification metrics is Confusion Matrix. It is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. It gives us an insight not only into the errors being made by the classifier but more importantly the types of errors that are being made, so for this part we must include confusion matrix as an important metric, Figure 7 shows the confusion matrices for each Neural Network that we applied, one before tuning and two others after tuning. There is also a classification report for each Neural Network in the code notebook, it's a text report showing the main classification metrics such as precision, recall and f1-score.

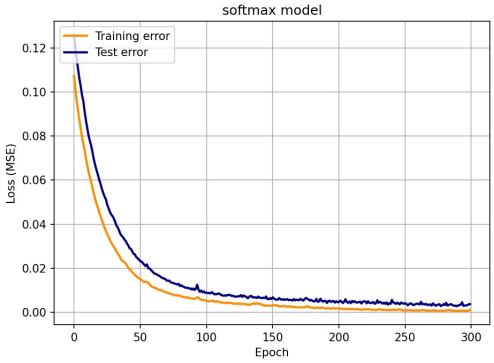
7 Tuning (Hyperparameter Optimization)

There are some Hyperparameters in the Neural Network like Activation functions in hidden layers, Number of nodes, Number of hidden layers and Learning rate for optimizer etc. We can tune these Hyper-parameters in order to get a better prediction. For Hyperparameter Optimization we used three techniques: Random Search optimization, Bayesian optimization and Hyperband optimization. The main process of what we did in the regression problem is the following: For Random Search optimization and Bayesian optimization we had some candidates for each hyperparameter, Number

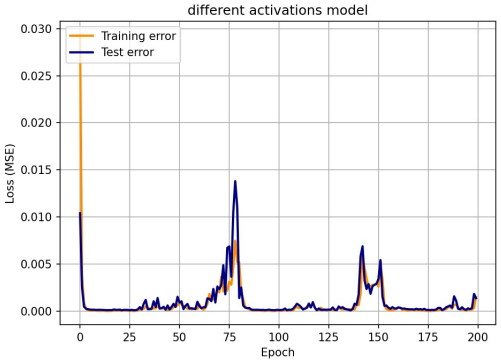


(a) Relu

(b) Tanh



(c) Softmax



(d) Mixed

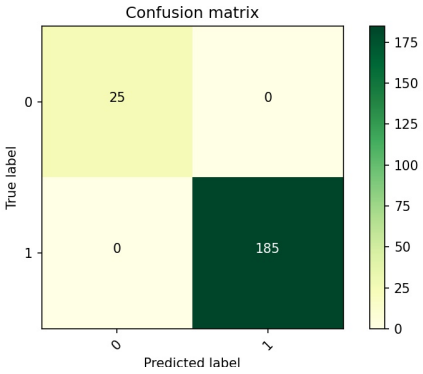
Figure 15: Loss vs. Epoch Curves for Regression

of hidden layers between 2 to 10, Number of nodes between 20 and 220, Learning rate to be 0.1 or 0.001 or 0.0001 and Activation functions to be relu, softmax and tanh. Then we decided to see what will happen if our Neural Network has only one hidden layer? We used Hyperband optimization to find the best Number of nodes for hidden layer and best Learning rate, we designed three single hidden layer Neural Networks with three different activation function: Relu, Tanh and Softmax, for these three Neural Networks we also used l2 loss for regularization. For the classification problem we followed the same precedure for Random Search optimization and Bayesian optimization, we did not use Hyperband optimization for classification part. Next we will explain briefly how these three optimization methodes work.

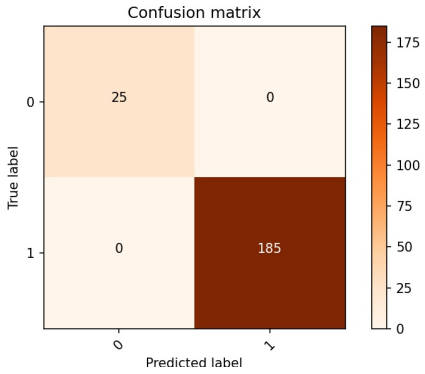
Random search [1] and Hyperband optimization [3] (a specific variation of random search) is a kind of optimization method that do not work with gradient and some well-known algorithms. The method is first select a random variable for cost function in the relative space and then choose another random variable and check that if the previous one is larger or not and by repeating this method, one may select the best value. But the Bayesian optimization [2] uses Bayesian rule for the base of the procedure. In a short description, it tries to minimize some cost function using prior distribution.

If we tune our neural network by applying these three optimizations, the loss vs. epoch curves for regression (Figure 8) show that for each one, there are some fluctuations impact on the loss function. But since these fluctuations are near zero, there is nothing problematic to conside.

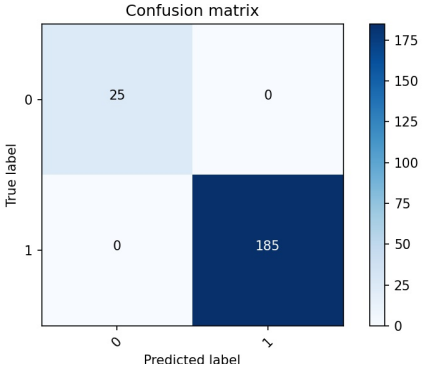
Finally, for regression problem we can compare the actual data with predicted values by our models after tuning. (Figure. 9, 10, 11, 12, 13)



(a) FNN after tuning with Random Search Opt.

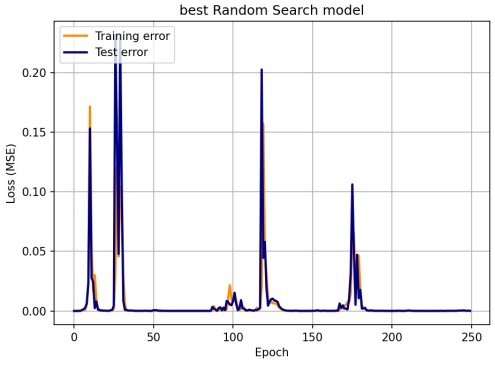


(b) FNN after tuning with Bayesian Opt

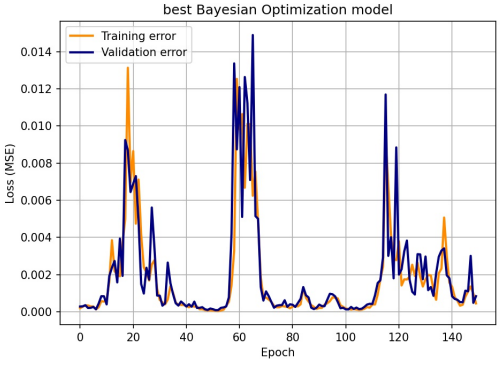


(c) FNN before tuning

Figure 16: Confusion matrices



(a) Random Search Optimization



(b) Bayesian Optimization

Figure 17: Loss vs. Epoch Curves for Regression

8 Conclusion

8.1 Traditional Techniques

We are looking for the best and fastest model of training. From the following table we can see that Gaussian Process Regressor is proper for fitting. As we can see although the fit and predict time of this model is very larger than the others but the error is about 6.90810^{-06} and so it is very optimal model.But if we want to consider fit time, we can refer to our learning curves, the third curve which shows performance of model (MSE vs fit times) helps us to find a model that has high score (small loss) and is also fast, we can see that Kernel Ridge Regressor has the best performance with respect to time. After that there are KNN Regressor and Decision tree Regressor, SVR is also a fine model but not optimal in compare to others, and as we expected Linear Regression with its extremely large error is not a proper model for our problem.

Model	MSE	Fit Time	Predict Time
KNN Regressor	4.386e-04	256 ms	146 ms
Decision Tree Regressor	3.238e-04	280 ms	6.98 ms
Gaussian Process Regressor	6.908e-06	3.59 s	1.3 s
Kernel Ridge Regressor	2.759e-05	99.9 ms	1.3 s
Support Vector Regressor	4.775e-03	169 ms	19.9 ms
Linear Regressor	4.777	179 ms	6.89 ms

8.2 Neural-Network

We are looking for the best and fastest neural network for our probelm. From the following table we can see that if we want to consider the overall error, the Neural Network which is tuned with Random Search Optimization is the best model. As we can see although the fit time of this model is larger than most of other models but the MSE is about $4.7497e-05$ and so it is very optimal Neural Network and it is as good as the best traditional models like GPR and KRR that we used in the privious part of this project. If we concentrate on the prediction of models for phase transition point (h_{PT}) which is one of our important aims, we can see that the model which is tuned by Bayesian optimization is the best for predicting this particular point because it is near to the exact theoretical value ($h = 1$). As we expected the Neural Networks with single hidden layer are not performing well in compare to others although they're tuned with Hyperband Optimizer.

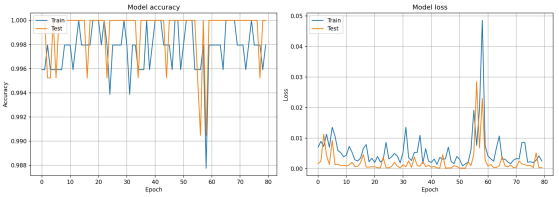
Feedforward Neural Network	MSE	h_{PT} Prediction	h_{PT} MAE	Fit time	Predict Time
All hidden layers Relu	1.22e-04	1.0086	8.6e-03	10.9 s	102 ms
All hidden layers Tanh	4.87e-03	1.0151	1.5e-02	10.2 s	102 ms
All hidden layers Softmax	3.79e-03	0.9882	1.1e-02	21.1 s	81.5 ms
Different hidden layers	1.39e-03	1.0114	1.1e-02	20.5 s	103 ms
Random Search Opt.	4.74e-05	0.9921	7.9e-03	55 s	48.4 ms
Bayesian Opt.	8.39e-04	1.0023	2.3e-03	13.2 s	95.2 ms
Hyperband Opt. (Relu)	2.13e-02	1.0305	3.05e-02	52 s	77.8 ms
Hyperband Opt. (Tanh)	7.73e-03	0.9541	4.5e-02	61 s	67.2 ms
Hyperband Opt. (Softmax)	2.13e-02	1.0306	3.06e-02	55 s	48.4 ms

The reason that some models gave us a better prediction for Phase transition point although they have larger MSE and vice versa is the fact that MSE is calculated by averaging over all 701 samples but Phase transition point is only one sample in our data and it may be predicted more precisely by a model with larger MSE. For general conclusion on models we think that the MSE is a much better metric to look at than the Phase transition point, since the Phase transition point (h_{PT}) maybe only accidentally predicted better in a particular run of the model while that model is not working very well on the overall regression problem of predicting h and determining phase of the system. One may ask, why we have this difference between the numerical and theoretical solutions? The answer is that in theoretical case, we assume that the number of atoms are infinite but in numerical way we considered only finite numbers of spins (12 atoms) and this leads us to this difference in our results.

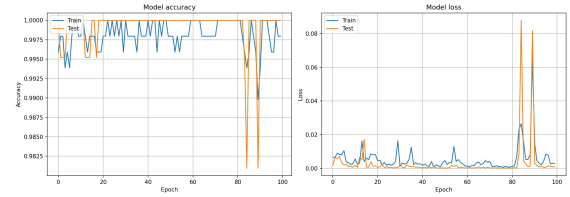
9 Related Works

Now we try to mention some papers that are relevant to our project.

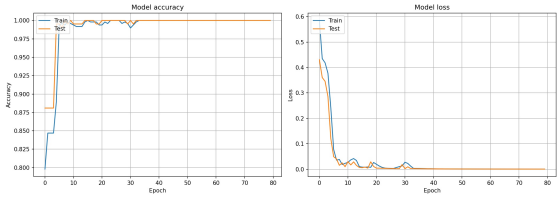
1. J. Carrasquilla and R. G. Melko "Machine learning phases of matter"



(a) FNN after tuning with Random Search Opt



(b) FNN after tuning with Bayesian Opt



(c) FNN before tuning

Figure 18: Loss& Accuracy vs Epoch Curve for Classification

In this article the authors try to use Machine Learning techniques to detect phase transitions of matter, especially for quantum ising model for first time. They showed that the Classification of phases occurs within the neural network without knowledge of the Hamiltonian or even the general locality of interactions. This article is the basis of our project with the difference that they worked on 2D square-lattice but we are working at 1D chain of 12 atoms.

2. C. Bauer and K. Hyatt "Quantum Ising Phase Transition"

Bauer and Hayatt in this article try to find the transition of phase in Ising model numerically. By the exact diagonalization of the Hamiltonian at first, they found the ground state wave function and then they checked out the effect of transverse magnetic field on overall magnetization and tried to find phase transition. We did a similar process to obtain our data.

3. F. D'Angelo1 and L. Böttcher "Learning the Ising model with generative neural networks"

The authors of this article trained the Ising model to machine with the technique of neural networks. They used Restricted Boltzmann machine (RBM) which is particular type of stochastic neural networks to learn the distribution of Ising configurations at different temperatures. This enabled them to examine the ability of RBMs to capture the temperature dependence of physical features such as energy, magnetization, and spin-spin correlations.

4. A. V. Uvarov, A. S. Kardashin and J. D. Biamonte "Machine Learning Phase Transitions with a Quantum Processor"

The above article tries to use Machine Learning phase transition as a tool to classify phases of matter with Quantum Processor. This article proposed a nearest-neighbour quantum neural network. This quantum classifier is successfully trained to recognize phases of matter with $\%99$ accuracy for the transverse field Ising model.

5. Han-qing Shi, Xiao-yue Sun and Ding-fang Zeng "Neural-network Quantum State of Transverse-field Ising Model"

In this article, they tried out neural-network quantum state (NQS) representation and machine learning method to reconstruct the ground state of the Transverse-field Ising Model (TFIM), both in one and two dimensions, and calculated its key observables, especially the entanglement entropy (EE). For the Stochastic Reconfiguration (SR method), they provided an understanding based on least action principle and information geometry.

6. Wei Zhang, Lei Wang and Ziqiang Wang "Interpretable Machine Learning Study of Many-Body Localization Transition in Disordered Quantum Ising Spin Chains"

They applied machine learning to the classification of two different phases, the eigenstate thermalization hypothesis (ETH) and the Many-body localization (MBL). They built and operated the support vector machine (SVM), designed for the random transverse field Ising chain. They demonstrated that the trained SVM with appropriate kernel choice is able to distinguish the two phases and determine the phase boundary. They trained data from two different energy

densities to make the trained SVM work for the whole energy spectrum. This fact ensures that during the training process, the models are built on properties of the MBL phase itself which should not depend on energy. They studied and understood how the SVM makes the decision. They found evidence that the SVM has the ability to automatically choose 2 a decision function which is very closely related to the many-body inverse participation ratio (IPR) defined in the configuration space.

7. Remmy Zen, Long My, Ryan Tan, Frederic Hebert4, Mario Gattobigio, Christian Miniatura, Dario Poletti and Stephane Bressan "Finding Quantum Critical Points with Neural-Network Quantum States"

They had an approach to finding the quantum critical points of the quantum Ising model using innate restricted Boltzmann machines (RBMs), transfer learning and unsupervised learning for neural-network quantum states (NQS). They showed that their approach can significantly improve the efficiency and effectiveness of a simple network like RBM.

10 Acknowledgments

Here, it is needed to many thanks to Dr. Abolhassan Vaezi as the supervisor of this project and our instructor of the "Machine Learning in Physics" course, Dr. Sadegh Raeisi.

References

[1] Juan Carrasquilla, Roger G. Melko, *Machine Learning Phases of Matter*. Nature Physics: Vol 13, p.431-434 (2017)

[2] Bikas K. Charkrabarti, Amit Dutta, Parongama Sen, *Quantum Ising Phases and Transition in Transverse Ising Models*. Lecture Notes in Physics: New Series m: Monographs, Springer (1996)

[3] Mehta, Pankaj, et al. *A High-Bias, Low-Variance Introduction to Machine Learning for Physicists*. Physics Reports: Vol 810, p.1-124 (2019)

[4] Zahra Mokhtari, *Non Equilibrium Dynamics of Quantum Ising Chains in the Presence of Transverse and Longitudinal Magnetic Fields* . Master Thesis, Simon Fraser University (2013)

[5] Alexander Weiße, Holger Fehske, *Exact Diagonalization Techniques*. Lecture Notes Physics, 739, 529–544, Springer (2008)

[6] Jonas Maziero, *Computational partial traces and reduced density matrices*. International Journal of Modern Physics C: Vol.28, No.01, 175005 (2017)

[7] M. Schumer, K. Steiglitz, *Adaptive Step Size Random Search*. IEEE Transactions on Automatic Control, Vol 13, Issue 3 (1968)

[8] Jonas Mockus, *Bayesian Approach to Global Optimization (Theory and Applications)*. Kluwer Academic Publisher (1989)

[9] Lisha Li, et. al., *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. Journal of Machine Learning Research, Vol 18, p.1-52 (2018)