Inverting the Ising Coupling Matrix Equation

Marina Drygala

2018/12/21

1 Introduction

The aim of the paper is to approximate an inverse equation for the Ising Coupling Matrix equation:

$$J_{i,j} = \sum_{n=1}^{N} \Omega_{i,n}\Omega_{j,n} \sum_{m=1}^{N} \frac{\eta_{i,m}\eta_{j,m}\omega_m}{\mu_n^2 - \omega_m^2} \tag{1}$$

We wish to approximate $J_{i,j}^{-1}$ as a function of Ω and μ. $J_{i,j}$, Ω and μ are of dimensions $N \times N$, $N \times m$, and m respectively. The values have a physical meaning, m refers to the number of detunings in the system and N the number of ions in the chain. For a more detailed description refer to [1].

2 Attempts at Solving the Problem

2.1 Attempt with μ fixed

The relevant notebooks for this are in the folder titled MuFixed.

We determined μ using the following equation:

$$\mu_m = \begin{cases} \omega_m + 0.1(\omega_{m+1} - \omega_m) & \text{if } m = 1, \dots, N-1 \\ 1.01\omega_m & \text{if m=N} \end{cases} \tag{2}$$

In some instances we experimented with setting $m < N$. To accomplish this we chose $S \subset 1, \dots, N$ and removed μ_s $\forall s \in S$ so that μ was of length m.

2.1.1 Data Creation

Consult the DataCreation file in addition to this document for a complete explanation.

The DataCreation file will generate $J_{i,j}, \Omega$ pairs to be used for training. Since each $J_{i,j}$ is a symmetric matrix with only zeros along the main diagonal, in most cases we will refer to the entries of the matrix that are above the main diagonal in vectorized form as being $J_{i,j}$. All $J_{i,j}$ values were normalized by dividing each entry by the L2 norm before training or any additional comparison was made.

To generate data the user must enter the number of ions in the chain, N, the trapping strength, and the number of examples they wish to generate before filtration.

Since we do not know which Ω, μ pairs will generate $J_{i,j}$ matrices that we are interested in, we impose that any training examples satisfy:

$$\Theta(J_{i,j}) > \epsilon \tag{3}$$

where:

$$\Theta(J_{i,j}) = \frac{1}{\frac{N(N-1)}{2}} \sum_{i>j} |J_{i,j}(i-j)| \tag{4}$$

and

$$\epsilon = \min_{\forall J_{i,j} \in T} \Theta(J_{i,j}) \tag{5}$$

and T is the test set, and contains the chain lattice, circular lattice as well as circular lattices with one additional interaction. The non-zero interactions in any $J_{i,j}$ entry of the test set are all of equal weight.

This filtration process is an inefficiency of this method, because a large amount of the data produced may not satisfy [3].

2.1.2 Training

Consult the TrainingNetworks file in addition to this document for a complete explanation.

The idea behind this paper is to approximate the inverse function using a neural network. Given training data generated as described above, the $J_{i,j}$ values are the network inputs and the Ω values are used as the network outputs. If a network is successfully trained it will take a desired normalized $J_{i,j}$ as input and produce a tensor Ω that can be used in the lab to produce approximately the desired lattice.

Network Architecture The network contains a fully connected layer, followed by ReLu activation, a 5% dropout layer, and two more fully connected layers.

The size of the input of the network is dependent on the size of the $J_{i,j}$ matrix. As previously mentioned we only need to consider the entries above the diagonal, which gives an input size of $\frac{N(N-1)}{2}$. The size of the hidden layers was set to be $128(N-2)$. As mentioned, the network outputs the predicted values of Ω, and so the size of the output is $N \times m$.

Other Network Details A batch size of 100 was used in training. The Adam Optimizer was used with the default learning rate. The loss function was Mean Squared Error with the true $J_{i,j}$ and the $J_{i,j}$ produced by the network output as the inputs. Since we are interested in producing a $J_{i,j}$ approximately equal to some desired value, we define the test error to be the mean of the percentage errors in the $J_{i,j}$ values in the test set:

$$\frac{1}{T} \times \frac{1}{\frac{N(N-1)}{2}} \sum_{t \in T} \sum_{\substack{i,j \\ i<j}} (J_{i,j}^{desired} - J_{i,j}^{predicted}) \times 100 \tag{6}$$

2.1.3 Deteriming Ω for a Desired Jij

Consult the PredictingOmega file in addition to this document for a complete explanation.

This notebook allows the user to predict the Rabi Frequencies, Ω, given a trained model. There is also code that tells the user which detunings contribute most to that particular $J_{i,j}$. This is useful as one can then experiment by generating new data after removing or modifying particular values of μ and then retraining.

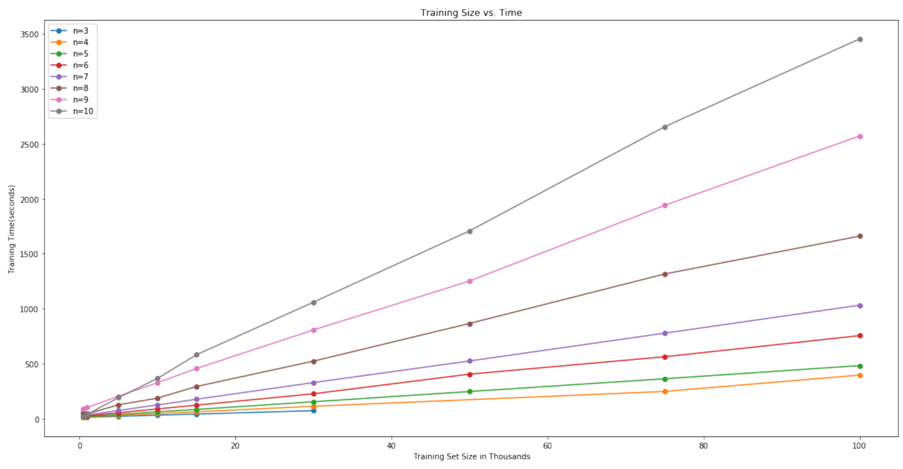

Figure 1: Shows the amount time required to train the network for 50 epochs.

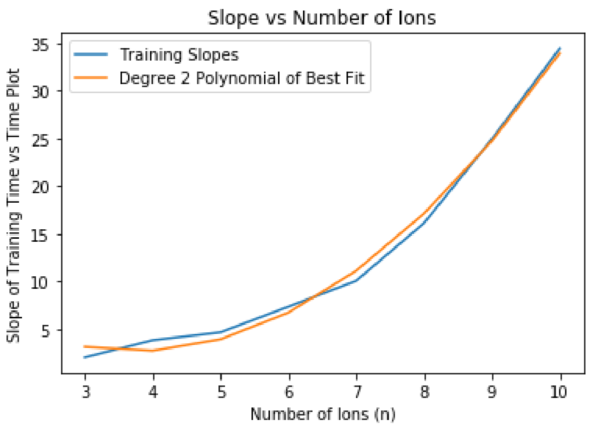

Figure 2: Slope of Graph that Compares Training Time and Training Size Plot for Different N

2.1.4 Results

Case 1: $m = N$ The network trains very well in the case where $m = N$. The training loss typically converges to a value less than 0.01, and the test error to well below 1%.

Figure [1] shows the relationship between the time required to complete a fixed number of epochs of training. For any given N there is a linear relationship between training set size and time. In Figure [2] we plot the slopes of these lines against N, and fit a polynomial of degree 2 to it. The polynomial of best fit is:

$$f(N) = 0.80225595N^2 - 6.03026786N + 14.02870833 \tag{7}$$

From Figure [3] we see that a training set size of 10,000 or greater leads to a reasonable error. This is feasible. Thus, we get that for a given N and a training set size, t_s we predict a training time of $f(N) \times t_s$ seconds.

Case 2: $m = N - 1$ Since additional detunings are costly to implement in the lab it is ideal to find a solution that minimizes m. One method explored of doing this was to observe which detunings were being used the least by a particular set of $J_{i,j}$ values. These detunings were then removed in order to

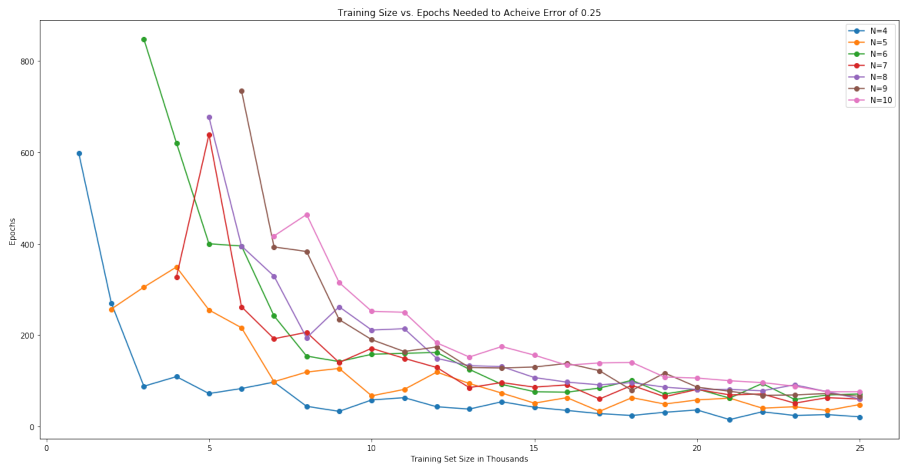

Figure 3: Comparison of the number of epochs required to achieve a test error of 0.25 for different N. One observation is that the result is unstable until the training set size is at least 10,000.

generate a new dataset and train a new network. This worked effectively for $m = N - 1$, but if this process was repeated to further reduce m it was ineffective.

Changing Cost Function Changing the inputs of the loss function to be the Ω and $\Omega^{prediction}$ results in a network that does not train. The implication of this is that using this network, it will be difficult to train the output parameters to fall in a desired range when μ is allowed to vary, and we wish to approximate it.

2.2 Attempts with μ Varying

The attempts to train a network with the same architecture and hyper-parameters modified so that the network predicts both Ω and μ were less successful.

Generating the values by adding Gaussian noise to the values of μ specified in [2] resulted in predictions of Ω that led to accurate $J_{i,j}$ values. However the values of μ that were predicted were not physical. Attempts to add a regularization parameter to limit the range of μ resulted in a network that did not converge.

3 Conclusion

One conclusion is that there appears to be multiple mathematical solutions to this inverse problem for a give set of parameters. This is concluded because $\Omega \sim U(-1, 1)$, yet the network predictions for Ω did not fall in this range. Similarly when μ was allowed to vary the network outputted values that did not fall in the range of those in the training set.

A second conclusion is that the the values for μ we see in [2] are good choices for the network and can be used to create lattices of interest. If we attempt to alter the values of μ aside from removing one entry we run into problems with this network structure and a new or alternative method will be required to solve this problem.

References

[1] S Korenblit,D Kafri, W C Campbell, R Islam, E E Edwards, Z-X Gong, G-D Lin, L-M Duan, J Kim, K Kim and C Monroe.(2012) *Quantum simulation of spin models on an arbitrary lattice with trapped ions.* New Journal of Physics.