# Inverting the Ising Coupling Matrix Equation

Marina Drygala

2018/12/21

## 1 Introduction

The aim of this 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 $\Omega$ and $\mu$. $J_{i,j}$, $\Omega$ and $\mu$ are of dimensions $N \times N$, $N \times m$, and $m$ respectively. The physical meaning of m and N, are the number of detunings in the system and number of ions in the chain. For a more detailed description refer to [1].

A project was completed attempting to use non-linear optimization to solve the problem. However this led to some undesirable properties, and so this paper focuses on using machine learning to approximate the inverse function.

## 2 Proposed Partial Solution

We find a good approximation for the function when $\mu$ is fixed as follows:

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

### 2.1 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 $J_{i,j}$.

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 $\Omega, \mu$ 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. It 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. This is because a large amount of the data produced may not satisfy [3].

All $J_{i,j}$ values were normalized by dividing each entry by the L2-norm before training or any additional comparison was made. In addition we normalize $\Omega$ by dividing each $\Omega_{i,n}$ by the square root of the L2-norm of the $J_{i,j}$ vector. We found the normalized pairs are the most efficient to train on.

## 2.2 Training

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

As previously stated, the motivation for this paper is to approximate the inverse function using a neural network. Given training data generated as described above, the normalized $J_{i,j}$ values are the network inputs and the normalized $\Omega$ 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 $\Omega$ that can be used in the lab to produce an approximation to the desired lattice.

### 2.2.1 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 $\Omega$, and so the size of the output is $N \times m$.

### 2.2.2 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:

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

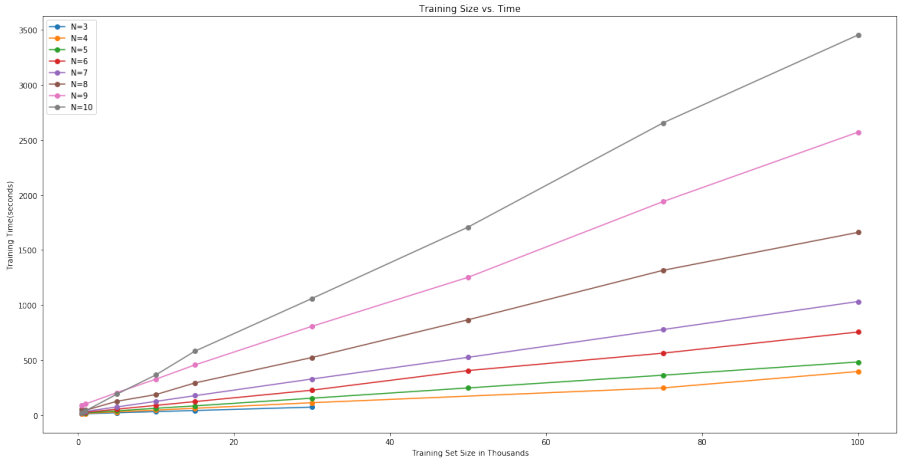


Figure 1: Shows the amount time required to train the network for 50 epochs.

## 2.3 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, necessary to approximate a desired $J_{i,j}$, given a trained model. A visualization of how the desired $J_{i,j}$ values compare is produced. There is also code that tells the user which detunings contribute most to that particular $J_{i,j}$.

## 2.4 Results

### 2.4.1 Computational Power

All of the computation was done on a 2015 MacBook Pro with a 2.2 GHz Intel Core i7 processor and 16 GB 1600 MHz DDR3 memory.

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 converges with stability. 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.

### 2.4.2 Accuracy

Given 25,000 training examples we trained very accurate networks for $N = 3, \ldots, 12$. We provide some examples of desired lattices, in Figure [4] as well as approximations given by our networks in Figure [5].

## 2.5 Other Pathways Explored and Future Possibilities

### 2.5.1 Altering $\mu$

Since additional detunings are costly to implement in the lab it is ideal to find a suitable solution that minimizes m.

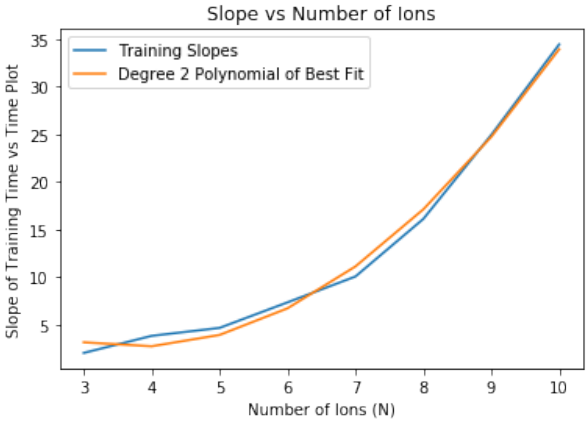


Figure 2: Slope of Graph that Compares Training Time and Training Size Plot for Different N

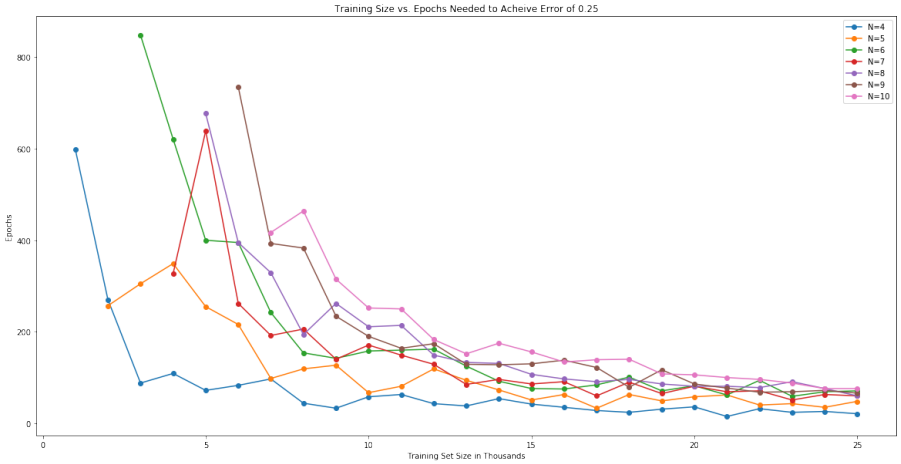


Figure 3: Comparison of the number of epochs required to achieve a test error of 0.25 for different N. One observation is that the result is unstable until the training set size is at least 10,000.

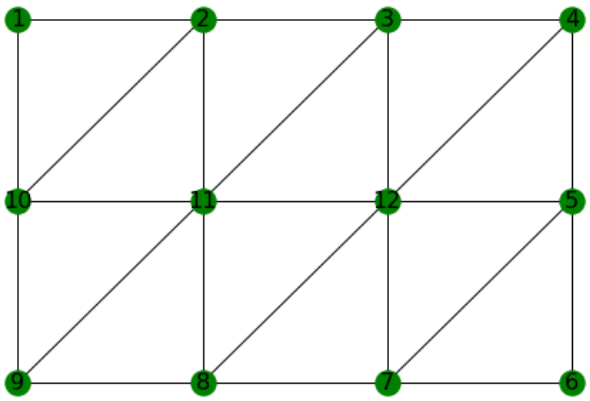


Figure 4: Triangular Lattice for N=12

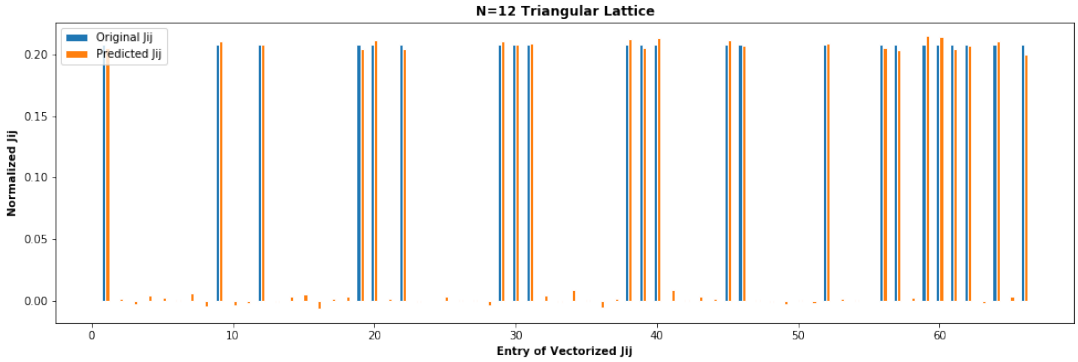


Figure 5: Approximation for Lattice in Figure [4]

**Removing values from Equation [2]** A first 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 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.

**Adding Gaussian Noise to Equation [2]** We trained the same network to predict both $\mu$ and $\Omega$ for N=4 by modifying only the output size. The values predicted for $\mu$ were not physical, and so although the test error was desirable more work in this area would be needed. We explore some possibilities on how to restrict the range of $\Omega$. This may be a useful starting point for restricting $\mu$.

**Changing Cost to Compare $\Omega$ Values** Changing the inputs of the loss function to be $\Omega$ and $\Omega^{predicted}$ results in a network that does not train. It is noted that the distribution for the values of $\Omega_{predicted}$ by the network do not match the distribution of $\Omega$ when a network is trained using the original cost function.

**Adding Regularization to $\Omega$** Altering the training code to include regularization for $\Omega$ converged to a desirable error when tested on for N=10 and lamda=0.001. This resulted in a distribution of $\Omega^{prediction}$ with a significantly reduced variance than without regularization.

The alterations made inside the training loop are given below.

```
model.train()
    for batch, (Jij, Omega) in enumerate(training_loader):
        optimizer.zero_grad()
        Omega_pred = model(Jij)
        Jij_pred = bsslmu(ic, mu, Omega_pred.view(batch_size,N,m), dev=device).normalize
        scaled_Omega_pred = lamda*Omega_pred
        x=torch.cat((Jij.double(),scaled_Omega_pred.double()),1)
        y=torch.cat((Jij_pred.double(),torch.zeros(scaled_Omega_pred.size(0),scaled_Omeg
        loss=criterion(x,y)
        loss.backward()
        optimizer.step()
    losses.append(loss.item())
```

# 3 Conclusion

The proposed network is useful at predicting values of $\Omega$ for the detunings given by Equation [2]. More work should be done to explore methods effectively for reducing the m, the number of detunings used.

# 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.