Solving the Bose-Hubbard model with Machine Learning

S. Gauthameshwar
National Institute of Science Education and Research

Abstract

This project primarily explores the field of quantum many-body physics with the help of machine learning. It gives the feel of how we can integrate physics and computer science concepts to solve a quantum many-body problem known to be NP-hard. I consider a Bose-Hubbard Hamiltonian that is easily implementable in an optical lattice system and predict the phase transition from a superfluid to a Mott-insulator. Borrowing the concepts of Feed-forward neural networks in machine learning, I perform a variational learning method to train the model and converge to a ground state. By taking a nine-site lattice with nine bosons, I find the ground state using feed-forward neural networks and see a good agreement with the state predicted by the DMRG algorithm. I have also obtained the signature of phase transition from the particle occupation of sites in the ground state. The theory of our phase transition, a summary of machine learning, and the details of the variational principles used to train our model are discussed.

The Bose-Hubbard model

The Bose-Hubbard model (BHM) is a model that describes the physics of discrete lattices with bosons. Although real life involves continuous systems both in space and time, we can still manage to capture many important physical properties that occur in condensed matter systems, such as superconductivity, electron spin scattering and local magnetism due to impurities, spin-orbit coupling, and transition from a conductor to insulator using the famous Hubbard model. Hubbard model works on a simple principle of on-site energy contribution and hopping energy contribution. Hopping happens only to neighboring sites, and the on-site potential includes a site-dependent potential of form $V(x_i)$. Also, it includes a chemical potential term if we allow freedom of changing particle numbers by going to the grand canonical ensemble. The interaction comes into play only if more than one particles occupy the same site. If we consider all these factors in a bosonic system, we end up with a Hamiltonian of the form:

$$\hat{H} = -J \sum_{\langle i,\ j\rangle} a_i^{\dagger} a_j + a_j^{\dagger} a_i \ + \ \sum_i (V_i - \mu)\, a_i^{\dagger} a_i + \sum_i \frac{U}{2} a_i^{\dagger} a_i \left(a_i^{\dagger} a_i \ - \ 1\right) \tag{1}$$

where $\langle i,\ j\rangle$ represents sum over neighbouring sites, J is the hopping strength, V_i is the on-site potential as a function of site index i , μ is the chemical potential, and $\frac{U}{2}$ is the interaction term that acts on all sites equally.

Discrete systems like ours can be experimentally realized in an optical lattice where we can create potentials where atoms can sit using lasers interference. These optical lattices usually operate when our atoms are cooled to ultra-cold temperatures, as light intensities cannot hold atoms if they have thermal energy to escape them. Our bosonic systems become a Bose-Einstein condensate at this regime. So we also get rid of all thermal noises, and we get a pure system in terms of noise. The only fluctuation existing in our bosonic system is quantum. Thus, everything happening in our system is an outcome of quantum mechanics. That is also the reason this specific transition from a superfluid to a Mott insulator is termed a *Quantum phase transition.*

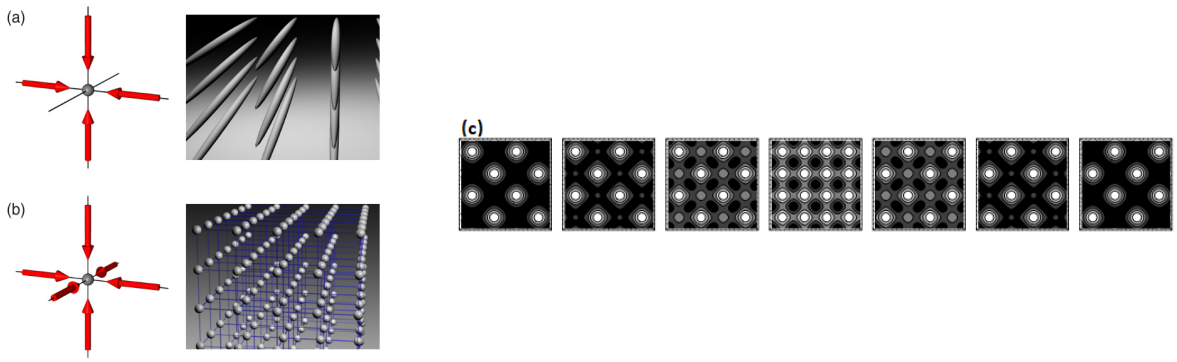


Fig.1 An optical lattice formed by superimposing counter-propagating, far red detuned laser beams superimposing. (a) shows a 2D array of lattice potential formed by two standing waves orthogonal to each other, (b) shows a 3D lattice potential can be created by superimposing three standing waves, and (c) shows how we can create check-board arrangement of potential with parallel polarized lights at different time phases of φ = 0^o, 30^o, 60^o, 90^o, 120^o, 150^o and 180^o respectively[1].

Having explained our experimental realization and model to simulate it, we customize the on-site potential to mimic the effect imparted in our physical system. We set the on-site potential of our system to:

$$V(i) = V\left(i - \frac{L}{2}\right)^2$$

Interfering lasers experience this potential in an experimental setup. We will work on a 9-site lattice with nine bosons having nearest neighbor hopping with periodic boundary conditions. Suppose we impose particle conservation (as we load a finite number of atoms in our optical lattice, and their number is kept fixed) and ignore the chemical potential effect. In that case, we end up with the final BHM as follows:

$$\hat{H} = -J \sum_{i=1}^{9} (a_i^\dagger a_{i+1} + a_{i+1}^\dagger a_i) + V\left(i - \frac{L}{2}\right)^2 \hat{n}_i + \frac{U}{2} \hat{n}_i(\hat{n}_i - 1) \tag{2}$$

Phase transition

1. Introduction to phase transition

I shall give a brief overview of the concepts of phase transitions that I'll be using to land on the superfluid to Mott insulator phase transition in section 2.

Phase transition in a system is when a parameter characterizing our system changes drastically for specific physical conditions. We see this every day around us. The entropy of the liquid gets a sharp discontinuity when it boils and becomes gas as we change the temperature of our system. Iron loses its magnetization when heated beyond the Curie temperature T_c. However, to characterize this phenomenon, we need a system parameter that encodes this physics. That is called the order parameter. There are two types of phase transitions depending on how this order parameter changes at the point of phase transition.

- First order phase transition

Here, the order parameter changes drastically and acquires a finite discontinuity. These phase transitions occur without warning and change the system abruptly. They also involve concepts of latent heat when they undergo the transition. Examples include condensation, boiling, melting, etc.

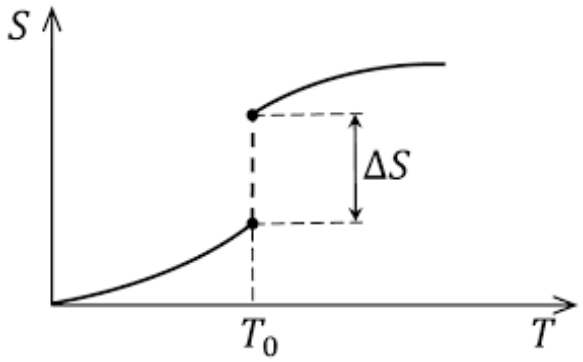


Fig. 2 An example of first order phase transition in a system undergoing liquid→gas transition. The order parameter is entropy *S*, the variable being changed is temperature and as the liquid gains a finite amount of entropy, it becomes a gas. Image courtesy: https://itp.uni-frankfurt.de/~gros/Vorlesungen/TD/6_Phase_transitions.pdf

- Second order phase transition

These are phase transitions that are continuous in the order parameter but has a jump in its rate of change. They give a warning to the observer before happening. Examples include ferromagnetic to paramagnetic transition of Iron when it is heated.

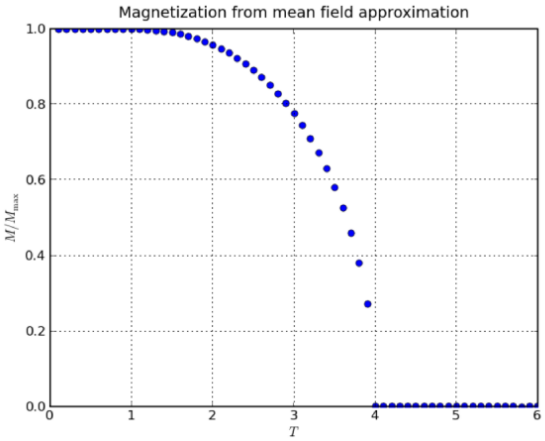


Fig. 3 An example of second order phase transition in a ferromagnet. The order parameter is magnetization $\overline{m}$, the variable being changed is temperature and as it goes from finite to zero, it becomes from a ferromagnet to paramagnet by losing all of its magnetization[2].

To theoretically formulate this phase transition, we use the Landau's theory to expand the system energy in terms of the order parameter and analyze its derivatives to obtain transition points.

Landau's theory for second order phase transition

We consider second order phase transition theory since our superfluid to Mott insulator is a second order phase transition.

We can describe the energy of our system as a function of our system as a power series expansion in terms of the order parameter ψ and system variable U as:

$$E(U, \psi) = E_0(U) + E_1(U)\,\psi + E_2(U)\,\psi^2 + E_3(U)\,\psi^3 + \ldots \tag{3}$$

The coefficients of each order is a function of the system variable. Our goal is to find the transition point U_c where the energy undergoes a second order phase transition.

A second order phase transition happens where the derivative of our energy w.r.t ψ is zero (i.e) when the below equation is satisfied:

$$\frac{\partial}{\partial \psi} E(\psi = \psi_c) = 0 \tag{4}$$

2. Phase transition in our system

To apply the Landau's theory to find the point of phase transition in our system, we first need to find the ground state energy of Eq.1 in terms of interaction, chemical potential and hopping strength. We can write non-dimensionalize the Hamiltonian by setting $J = 1$, and considering all other terms in the units of hopping strength. So we have the energy of our system as $E_g(\mu/J,\ U/J,\ \psi)$. Now to find the ground state, we apply mean field approximation:

$$\hat{a}_i^\dagger \hat{a}_j = \langle \hat{a}_i^\dagger \rangle \hat{a}_j + \hat{a}_i^\dagger \langle \hat{a}_j \rangle - \langle \hat{a}_i^\dagger \rangle \langle \hat{a}_j \rangle \tag{5}$$

to the hopping part of the Hamiltonian and consider it as a perturbation to our on-site and interaction Hamiltonian.

$$\hat{H} = \sum_i (V_i - \mu)\,\hat{n}_i + \sum_i \frac{U}{2}\hat{n}_i(\hat{n}_i - 1) - J \sum_{\langle i,j \rangle} a_i^\dagger a_j + a_j^\dagger a_i = H_0 + H_{\text{hop}} \tag{6}$$

where $H_0 = \sum_i (V_i - \mu)\,\hat{n}_i + \sum_i \frac{U}{2}\hat{n}_i(\hat{n}_i - 1),\ H_{\text{hop}} = -J \sum_{\langle i,j \rangle} a_i^\dagger a_j + a_j^\dagger a_i$

Why consider hopping as perturbation? Because it turns out the unperturbed Hamiltonian H_0 has a neat eigenstates which are the position basis themselves in the Fock space.

Simplifying Eq. 6 by using Eq. 5, we get:

$$\hat{H} = \frac{U}{2}\sum_i \hat{n}_i(\hat{n}_i - 1) - \mu \sum_i \hat{n}_i - t \sum_i \left(\psi^* a_i + \psi\, a^\dagger{}_i - |\psi|^2\right) \tag{7}$$

where $\psi = \left\langle a_i^\dagger \right\rangle$ is the expectation value of the creation operator.

It turns out that the ground state of H_0 has its lowest eigenstate corresponding to the occupation basis where

$$n_i = 1 + \left\lfloor \frac{\mu}{U} \right\rfloor \forall\ i. \tag{8}$$

To know the derivations of this in detail, please see section 14.1 of this chapter.

Now we apply the concepts of higher order perturbation theory to expand our E_g to first two non-vanishing terms. Unfortunately, it turns out this a pain-sticking work since we have to go to four orders of ψ to get what we want. After a lot of blackboard calculations we arrive at an expansion of $E_g(\psi)$ as something that gets ever monstrous as we go to higher terms:

$$\begin{aligned} E_g = {} & (V - \mu)\,n + \frac{U}{2}n(n-1) + \left(1 + \left(\frac{n}{U(n-1) - \mu} + \frac{n+1}{\mu - U n}\right)\right)\psi^2 + \\ & \left(\frac{(n+1)(n+2)}{(\mu - U n)^2 (2\mu - U(2n+1))} + \frac{n(n+1)}{(U(n-1) - \mu)^2 (U(2n-3) - 2\mu)} - \right. \\ & \left. \left(\frac{n}{U(n-1) - \mu} + \frac{n+1}{\mu - U n}\right)\left(\frac{n}{(U(n-1) - \mu)^2} + \frac{n+1}{(\mu - U n)^2}\right)\right)\psi^4 + O\left(\psi^6\right) \end{aligned} \tag{9}$$

here n is the number of bosons in our lattice.

The good news is that we are now able to find the coefficients that can now simplify our E_g as:

$$E_g(U,\ \mu,\ \psi) = E_0(\mu,\ U) + E_2(\mu,\ U)\,\psi^2 + E_4(\mu,\ U)\,\psi^4 + O\left(\psi^6\right) \tag{10}$$

where

$$E_2(\mu,\ U) = 1 + \left(\frac{n}{U(n-1) - \mu} + \frac{n+1}{\mu - U n}\right) \tag{11}$$

$$\begin{aligned} E_4(\mu,\ U) = {} & \frac{(n+1)(n+2)}{(\mu - U n)^2 (2\mu - U(2n+1))} + \\ & \frac{n(n+1)}{(U(n-1) - \mu)^2 (U(2n-3) - 2\mu)} - \left(\frac{n}{U(n-1) - \mu} + \frac{n+1}{\mu - U n}\right)\left(\frac{n}{(U(n-1) - \mu)^2} + \frac{n+1}{(\mu - U n)^2}\right) \end{aligned} \tag{12}$$

Now, we simply differentiate Eq. 10 w.r.t ψ as in Eq. 4 to get:

$$\psi_c = 0,\ \ |\psi_c|^2 = \frac{-E_2}{2E_4} \tag{13}$$

Only one of these solution is feasible depending on which regime of E_2 we are talking about. ψ_c being negative makes no physical sense as creation operators always produce a positive coefficient when acting on a Fock state. Hence, if $E_2 < 0$ $|\psi_c| = \sqrt{\frac{-E_2}{2E_4}}$ is the valid solution and if $E_2 > 0$, $\psi_c = 0$ is the valid solution. We also must notice that the second derivative of E_g at these ψ_c has a positive value, meaning they all minimize the ground state energy. All in consistent with physicality so far.

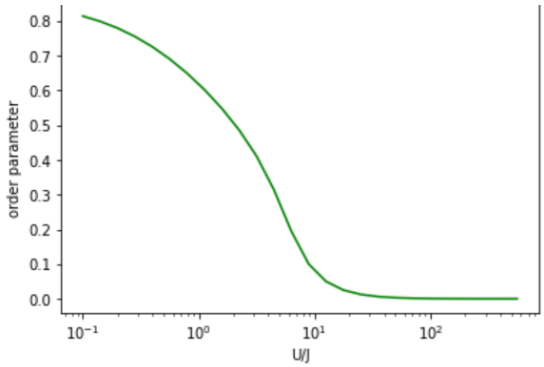


Fig. 4 Order parameter value for different values of U/J. $\mu = 0$, and we see a second order phase transition at $\frac{U}{J} \sim 10^1$. Order parameter simulation done for a boson lattice of 7 sites with 7 bosons.

If we go ahead and plot the mean field diagram of $\psi(\mu,\ U)$ for our BHM, we see patches of area where we do not have any creation of particles. So if we look at the grand canonical ensemble picture of our BHM, the system does not like to accept/give any particles. This is because there is a band gap existing between the ground state and the first excited state that makes it energetically infeasible. Thus, we have an insulator phase at the areas where $\psi_c = 0$, We have also seen that these insulating phases have integer occupation in Eq. 8. So our system has integer occupation of particles at each site. This is a Mott insulator!

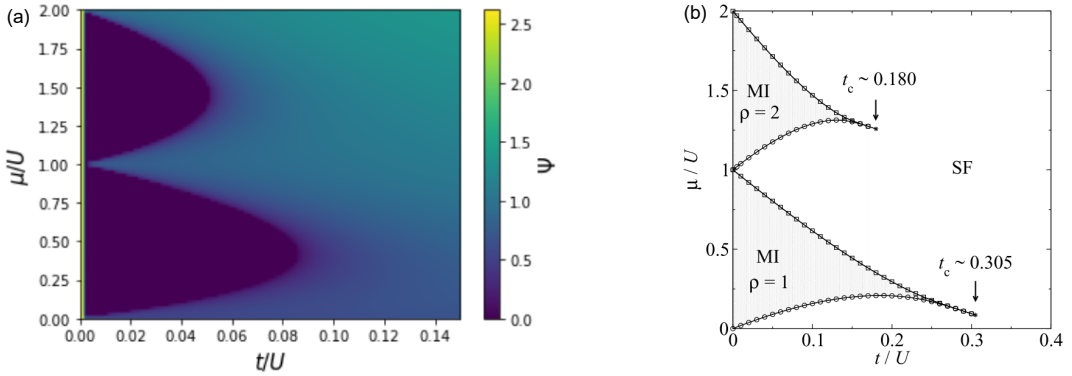


Fig. 5 Mean field phase plots for the Bose-Hubbard model. We consider the dimensional form of BHM by assuming a value for hopping strength *t*. (a) Heat-map of ψ for various values of $\frac{\mu}{U}$ and $\frac{t}{U}$ for $L = 9$[5]. (b) Mott lobes with boundaries defined inside which $\psi = 0$ for $L = 128$. t_c represents the point where the lobes close[4]

We have already seen in the class for the regime where $U \to 0$. Here, we get a tight-binding Hamiltonian that has a cosine energy distribution in the momentum space defined as:

$$\epsilon_k = -2\,t\,\mathrm{Cos}(k) + (V - \mu) \tag{14}$$

This system has its ground state corresponding to $k = 0$ as:

$$|\psi_g\rangle = \frac{1}{m!}\left(\sum_{i=1}^{N} \hat{a}_i^{\dagger}\right)^m \Big|\,\Omega\Big\rangle \tag{15}$$

This is a superfluid state since bosons are delocalized all over the lattice. Superfluid state has a continuous band structure and so, we have non-zero ψ characterizing systems in that state.

Machine learning

What is machine learning? There are several definitions of this, depending on how it is used. However, in our physical system of interest, machine learning defines a set of variables that the machine fixes in training to make a general prediction for a given input with a fair amount of generalization. If that is too much to digest, consider a simple example that helps illustrate the above statement. Consider a student, me, learning many body physics course. I am given some resources such as notes and lectures where I am introduced to how to use those notes and see how much I can apply them in an actual research problem. You cannot tell if I have learned the subject if you evaluate me from questions directly from the notes. I could as well memorize it without 'learning' it. A better solution might be to ask questions that are not directly from the notes but *based* on the notes and evaluate me (like making me do a many-body

project that uses the concepts in the class and evaluate my presentation/report). If I perform well, you are confident that I have learned many-body physics and give me AA. If we draw this analogy to a machine, the training set takes the role of notes/lecture materials that the machine uses to learn and fix its internal parameters. It uses the parameters to answer some questions as inputs the machine has not encountered while training, and the score the machine gets becomes the criteria of how well it has learned.

This simple concept bridges the link between the fast computation power of the machine and generalization to come up with solutions for problems the machine has never encountered before, just as humans can. This has motivated computer scientists to develop several ML models for training machines such as regression, clustering algorithms, support vector machines, random forests, and neural networks. Since I shall be using neural networks extensively in this project, I will focus on that.

Types of learning

Before jumping into neural networks I shall give a brief overview of the various learning methods quantum physicists use to solve their models so that we have a broad idea of the applications of ML in physics.

■ Supervised learning

Supervised learning is the method where every training instance has a correct answer, and the machine's goal in training is to reach that answer as close as possible for all the test instances. The correct answers are the *labels* for the given instances. Since we already have the answer in our hand, we involve the concept of a loss function $L(x_i, \bar{x}_i)$ to quantify the performance/error of our prediction, and then implement a correction scheme to change the machine parameters to ensure it reduces the quantified loss next time. Algorithms like neural networks, restricted Boltzmann machines, and random forests extensively use this to find the optimal wavefunctions of the model we wish to solve.

■ Unsupervised learning

Unsupervised learning is a method with no label to our training set. We have some features and are to infer some properties from them. Common unsupervised learning instances include clustering using k-nearest neighbors or plotting a probability or statistical distribution from our inputs. In quantum physics, unsupervised learning is used to learn the unknown probability distributions of our system. Such a process is achieved through quantum tomography, where we use repeated projections to infer the state of interest. Given an unknown quantum state, a neural network can be trained on projective measurement data to discover an approximate reconstruction of the state. You can watch this lecture if you are interested in knowing more about unsupervised machine learning for many-body physics.

Neural networks: The inspiration from nature

To address how to make a machine model capable of learning instances and answering them, computer scientists had to take a step back first to understand how we learn so many things as we encounter situations. Nature has evolved a delicate and complex mechanism to store information in us. That is a neural network. At its heart, it is just a simple concept implemented on a large scale. It is made of neurons that receive some input electric impulses from the other neurons it is connected. It then adds them all up and fires its signal with a non-linear response depending on the inputs it receives. We have a neural network if we take several such neurons and interconnect them! The non-linear response ensures that specific neurons only fire when provoked, and an avalanche of firing does not happen. This network has helped humans remember much information, from muscle memory to thoughts and experiences. Therefore, if the same neural network can learn and store information based on what the sensory organs receive, why won't it work for a machine if we mimic the system to train it as per our needs? This led to the breakthrough where we successfully made machines store information and implemented powerful training methods to optimize it to answer instances it had never encountered accurately.

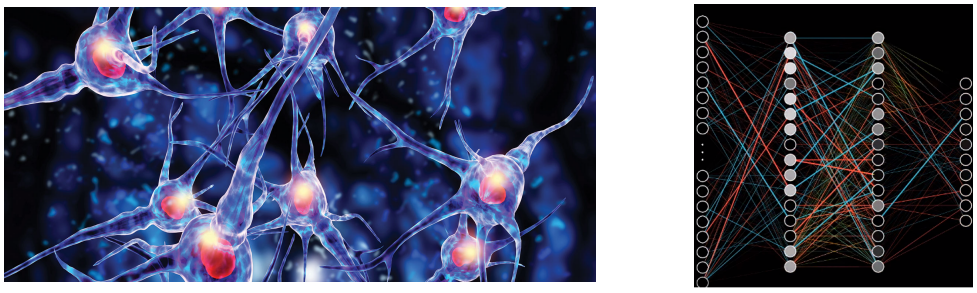


Fig. 6 (a) A depiction of neural networks in biological systems. (b) Schematic of an artificial neural network model used by machines

■ Feed-Forward Neural Networks (FFNN)

Feed-forward neural networks are a class of neural networks that has an input layer where the neurons receive information about the input, transmit it to a hidden layer, and then convolve it to an output of neurons which will give us the final values, hence their name.

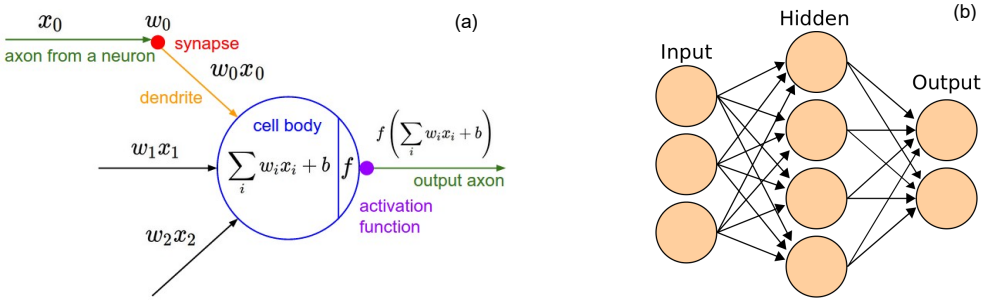


Fig. 7 (a) A functioning of a single neuron (also called as perceptron) in a FFNN. It takes input from its preceding neurons, scales each of them with a weight, adds its own bias to it, and fires a nonlinear activation from the calculated value. (b) shows several such neurons interconnected to form three layers of a neural network (that's why FFNN are also called Multi-Layered Perceptron).

Neural networks are one-shot learners, meaning they learn from just one instance. This differs from other training methods like regression or clustering, where we need information from all training instances to develop a generalization. However, we need several such shots to make our network an accurate model. These are four things a network does to train itself in one shot:

1. Take an input
2. Process the input of each neuron by applying the corresponding weights and fire a nonlinear response
3. Look how good the prediction is to an actual value
4. Do a back propagation to tweak the weights in the right direction

Let us say our input has n features, the hidden layer has m neurons, and the output has two neurons. Feed forward neural networks get an array of features $\boldsymbol{u}^{(0)} = \left(u_1{}^{(0)},\ u_2{}^{(0)},\ ...,\ u_n{}^{(0)}\right)$ as an input to the neurons in layer 0 and send it to the neurons in the proceeding layer 1 without any activation. The input a neuron i receives in layer 1 after applying the corresponding weights and biases is:

$$u_i{}^{(1)} = \sum_{j=1}^{n} w_{i\,j}{}^{(1)}\, u_j{}^{(0)} + b_i{}^{(1)} \qquad \forall\ i \in \text{hidden layer} \tag{16}$$

There are n weights for each neuron in the hidden layer since there are n inputs, and each neuron has its own bias b_i. The superscript represents the layer to which these weights and biases correspond to. This can be concisely written in a matrix form as

$$\begin{pmatrix} u_1{}^{(1)} \\ u_2{}^{(1)} \\ \dots \\ u_m{}^{(1)} \end{pmatrix} = \begin{pmatrix} w_{11}{}^{(1)} & w_{12}{}^{(1)} & w_{13}{}^{(1)} & \dots & w_{1\,n}{}^{(1)} \\ w_{21}{}^{(1)} & w_{22}{}^{(1)} & \dots & \dots & w_{2\,n}{}^{(1)} \\ w_{31}{}^{(1)} & \dots & \dots & \dots & w_{3\,n}{}^{(1)} \\ \dots & \dots & \dots & \dots & \dots \\ w_{m\,1}{}^{(1)} & w_{m\,2}{}^{(1)} & w_{m\,3}{}^{(1)} & \dots & w_{m\,n}{}^{(1)} \end{pmatrix} \begin{pmatrix} u_1{}^{(0)} \\ u_2{}^{(0)} \\ u_3{}^{(0)} \\ \dots \\ u_n{}^{(0)} \end{pmatrix} + \begin{pmatrix} b_1{}^{(1)} \\ b_2{}^{(1)} \\ b_3{}^{(1)} \\ \dots \\ b_n{}^{(1)} \end{pmatrix} \tag{17}$$

or even more concisely as :

$$\boldsymbol{u}_i{}^{(1)} = \boldsymbol{W}_{i\,j}{}^{(1)}\, \boldsymbol{u}_j{}^{(0)} + \boldsymbol{b}_i{}^{(1)} \tag{18}$$

In layer 2, the output neurons then receive the input after applying weights and biases to the hidden neuron's output as:

$$\begin{pmatrix} u_1^{(2)} \\ u_2^{(2)} \end{pmatrix} = \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} & \dots & w_{1m}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & \dots & w_{2m}^{(2)} \end{pmatrix} \begin{pmatrix} u_1^{(1)} \\ u_2^{(1)} \\ \dots \\ u_m^{(1)} \end{pmatrix} + \begin{pmatrix} b_1^{(2)} \\ b_2^{(2)} \end{pmatrix}$$

or in other words,

$$\boldsymbol{u}_i^{(2)} = \boldsymbol{W}_{ij}^{(2)} \boldsymbol{u}_j^{(1)} + \boldsymbol{b}_i^{(2)}$$

We then apply a nonlinear activation $\sigma(x)$ to $\boldsymbol{u}^{(2)}$ vector and obtain the output as:

$$\boldsymbol{v}_i = \sigma\left(\boldsymbol{u}_i^{(2)}\right) \tag{19}$$

Now that we have a linear algebra description of our FFNN, it all boils down to fixing the weights and biases by tweaking them in such a way that the machine is able to accurately predict our output. There are many ways to reach the optimum configuration, given the error such as SGD, AdaMax, AMSGrad, and RMSProp[5]. One simple and commonly used method is to follow a gradient descent. We assume the direction in which the gradient of our error w.r.t w_i points is the place where our weights must change to lower the overall error of the model and reach an equilibrium. Hence, once we calculate the cost function (the error) between our predicted value $\boldsymbol{v}$, and the actual value $\boldsymbol{v}^*$ as a function of the weights and biases: $L(w_1, w_2, \dots w_{mn}, b_1, b_2, \dots, b_m)$, we update each of our bias as:

$$w_i \leftarrow w_i - \gamma \frac{\partial L(\boldsymbol{W}, \boldsymbol{b})}{\partial w_i} \tag{20}$$

where γ is the learning rate that determines how fast we descend to the minima. This is called back-propagation. The FFNN when initialized will give a garbage result far from the actual value. But the idea is, if we train it multiple times by giving enough instances, the network converges to a stable configuration and represents our model as accurately as possible.

■ FFNN for our BHM

Now to our problem, we wish to find the ground state of a given Hamiltonian. Since this state exists in the Fock space represented by integer number occupation at each site, we train a FFNN that predicts the coefficient of each Fock state basis in the ground state. Clearly our output will be a complex number, so we have two outputs from the machine that will give us the real and imaginary part of our desired state. The two outputs are of the form: Re(ln(ψ)), and Im(ln(ψ)). Log(ψ) is taken for calculation convenience while computing the corrections to the weights and biases during back-propagation. The schematic of the FFNN is given in Fig. 8.

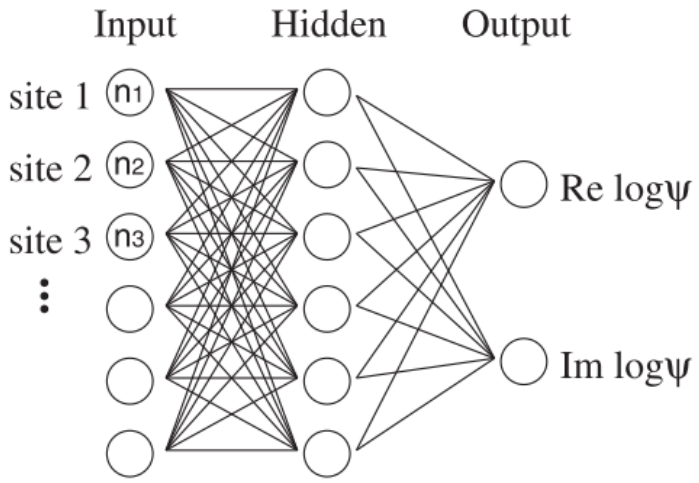


Fig. 8 Schematic diagram of the artificial neural network used to solve the Bose–Hubbard model. The number of particles at each site is assigned to the input layer, and the corresponding value of the wave function is obtained from the output layer. The units in the adjacent layers are fully connected.

If we have a bosonic lattice with 9 sites and 9 bosons, the input Fock states can be $|\boldsymbol{n}\rangle = |0, 0, 0, 0, 0, 0, 0, 0, 9\rangle$, $|1, 0, 0, 0, 0, 1, 2, 5\rangle$, etc. We represent the coefficient of any such $|\boldsymbol{n}\rangle$ in the Fock space as $\psi(\boldsymbol{n})$ where $\boldsymbol{n} \in \left\{\boldsymbol{n} \,\middle|\, \sum_{i=1}^{9} n_i = 9\right\}$. So our ground state becomes:

$$|\psi_g\rangle = \sum_{\boldsymbol{n}} \psi(\boldsymbol{n}) \left|\boldsymbol{n}\right\rangle \tag{21}$$

I keep 20 hidden neurons in the hidden layer (i.e), $m = 20$. Our FFNN has the input processing as described in Eq. 16-19. Hence, the

coefficient is related to the FFNN output as:

$$\psi(\boldsymbol{n}) = e^{u_1{}^{(2)}(\boldsymbol{n}) + i\, u_1{}^{(2)}(\boldsymbol{n})} \tag{22}$$

In Eq. 19, our FFNN has a tanh activation function $\sigma(x) = \tanh(x)$.

If I use supervised learning, I must create labels of the instances, which are the exact coefficients of the ground state energy. We now end up in a chicken-egg situation where we need to know the ground state energy to find the ground state energy, and using machine learning in instances where we have already information about the ground state seems redundant. Therefore, we take a variational approach to train our FFNN where we use Monte Carlo method to randomly pick a finite number of Fock state samples and implement Metropolis sampling to get a probability distribution of states that closely match that of the actual ground state. such variational methods are widely used to train many ML models such as CNN, RBM, FFNN, etc[6].

Variational method to train our model

■ Monte Carlo method and Metropolis sampling

Let us say the ground state has a probability distribution for each $|\boldsymbol{n}\rangle$. The ground state energy is given by the formula

$$\langle \hat{H} \rangle_{gs} = \sum_{\boldsymbol{n}} p_{gs}(\boldsymbol{n})\, E_{\boldsymbol{n}} = \frac{\sum_{\boldsymbol{n},\boldsymbol{n}'} \psi^*(\boldsymbol{n}) \left\langle \boldsymbol{n} \mid \hat{H} \mid \boldsymbol{n}' \right\rangle \psi(\boldsymbol{n}')}{\sum_{\boldsymbol{n}} |\psi(\boldsymbol{n})|^2} \tag{23}$$

Now, if our Hilbert space $\mathcal{H}$ is very large, it will not be possible to find this exact sum by summing over all the Fock states. Therefore, we consider a fairly large, but not exponentially large subspace $\mathcal{H}_{mc}$ of $\mathcal{H}$ and try to compute this average in $\mathcal{H}_{mc}$. This subspace is chosen by random pooling with replacement of Fock states from $\mathcal{H}$ and this process is called Monte Carlo method in our variational problem. Clearly, the probability distribution of $\mathcal{H}_{mc}$ won't be anywhere close to $p_{gs}(\boldsymbol{n})$, just from our random pooling. So we use the Metropolis sampling to define a probability distribution with which it picks a state in $\mathcal{H}_{mc}$.

Metropolis algorithm is an accept reject algorithm that tells how likely our system will be, to have a given state in its optimal configuration. If we pick a state $|\boldsymbol{n}\rangle$ from $\mathcal{H}_{mc}$ at random, and find out the probability of it changing to $|\boldsymbol{n}'\rangle$, that is given by:

$$p(\boldsymbol{n} \to \boldsymbol{n}') = \left(\left| \frac{\psi(\boldsymbol{n}')}{\psi(\boldsymbol{n})} \right| \right)^2 \tag{24}$$

If this probability is greater than 1, we are fully certain that our stable ground state will have a probability distribution where there is more $\boldsymbol{n}'$ than $\boldsymbol{n}$. Thus, we accept this state with probability 1. But what if it is less than 1? Do we reject it as it leads to an unstable configuration? *No*! We accept the new state with a probability $p(\boldsymbol{n} \to \boldsymbol{n}')$. After doing this iteration several times, we achieve a $\mathcal{H}_{mc}$ that has the characteristic probability distribution of our original $|\psi_{gs}\rangle$. Thus, we can now calculate the average ground state energy by using Eq. 23 and the summation indices now run over $\boldsymbol{n}$, $\boldsymbol{n}' \epsilon \mathcal{H}_{mc}$. Since our matrix $\hat{H}$ is sparse, we can also easily compute the coefficient $\left\langle \boldsymbol{n} \mid \hat{H} \mid \boldsymbol{n}' \right\rangle$ without much computational difficulty.

■ Short discussion on why we accept with a probability

The first intuition that strikes us when we see a transition from $\boldsymbol{n} \to \boldsymbol{n}'$ is less than 1 is that this transition will take our system to a configuration that is higher in energy than that when we remain with the old configuration. If we reject unstable configurations with full probability, we will be doing what is called a gradient-based minimization where we are extremely prone to get stuck at a local minima. Fig. 9 (a) illustrates this effect. However, if we allow for our system to reach unstable configurations with a small probability depending on how drastically the energy changes while choosing that transition, we allow that freedom to overcome local minima and descend down to a global and more stable one as shown in Fig. 9 (b). This in fact happens in physical systems in nature. The entropy of our system causes thermal fluctuations to happen and have finite probabilities to exist in higher energy states as well. In that case, Eq. 24 becomes $p(\boldsymbol{n} \to \boldsymbol{n}') = \left| e^{-\beta (E(\boldsymbol{n}') - E(\boldsymbol{n}))} \right|$. For our BHM that has no thermal noise, it is the quantum fluctuations that provide these finite probabilities to go to a higher energy configuration state. So Metropolis sampling is a perfectly physical and reasonable algorithm to follow.

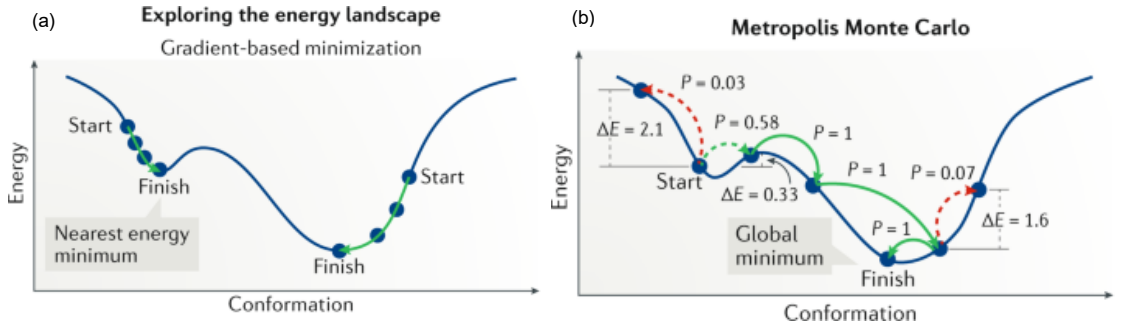


Fig. 9 (a) shows a gradient descent where there is a possibility of getting stuck in local minima as we do not allow our system to go to a higher energy configuration. x-axis represents the elements in our configuration space. (b) shows how we still accept a higher energy configuration with a probability to overcome local minima and reach a global minima.

■ Training our model

We proceed to do the back-propagation of our neural network by obtaining the expectation of our Hamiltonian over a Monte Carlo set, and use it to find the gradient. The coefficients of the states $\psi(\boldsymbol{n})$ in our variation algorithm is computed by our neural network. With that, it predicts $\langle \hat{H} \rangle_{mc}$ and uses it as the loss function to correct the weights as followed in Eq. 20. So the SGD correction for our FFNN after the appropriate differentiation becomes:

$$\frac{\partial \langle \hat{H} \rangle_{mc}}{\partial w} = 2\,\mathrm{Re}\left(\frac{\sum_{\boldsymbol{n},\boldsymbol{n}'} O_w^*(\boldsymbol{n})\,\psi^*(\boldsymbol{n})\,\langle \boldsymbol{n} \mid \hat{H} \mid \boldsymbol{n}' \rangle\,\psi(\boldsymbol{n}')}{\sum_{\boldsymbol{n}} \mid \psi(\boldsymbol{n}) \mid^2} - \langle \hat{H} \rangle_{mc} \frac{\sum_{\boldsymbol{n}} O_w^*(\boldsymbol{n}) \mid \psi(\boldsymbol{n}) \mid^2}{\sum_{\boldsymbol{n}} \mid \psi(\boldsymbol{n}) \mid^2} \right) = 2\,\mathrm{Re}\left(\langle O_w^* \hat{H} \rangle_{mc} - \langle O_w^* \rangle_{mc} \langle \hat{H} \rangle_{mc} \right) \tag{25}$$

where

$$O_w(\boldsymbol{n}) = \frac{1}{\psi(n)} \frac{\partial \psi(\boldsymbol{n})}{\partial w} \tag{26}$$

Assuming logarithmic output in our FFNN has given us a neat-looking SGD correction that our machine can implement to reach its optimum set of weights and biases.

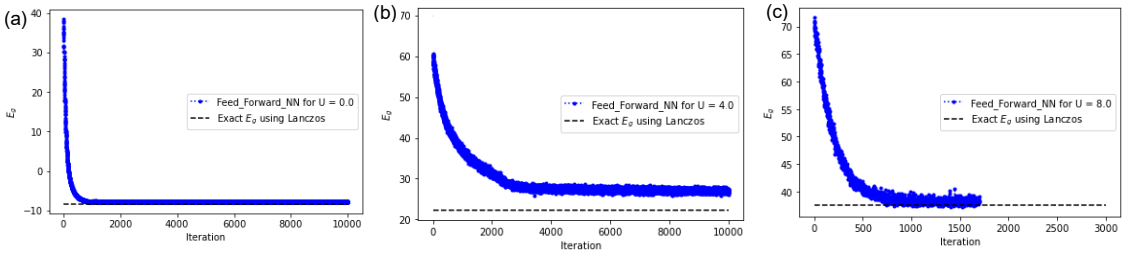
Numerical results

I shall implement a FFNN to solve the BHM of a 9-site lattice with 9 bosons. The no.of Fock states of a bosonic system with M sites and N particles is given as:

$$|\mathcal{H}| = \frac{(N+M-1)!}{N!\,(M-1!)} \tag{27}$$

For our case, using Eq. 27, the system size becomes 24,310. From this, we sample 1000 states for our variational computation. We define a maximum bosonic occupation $n_{\max} = 6$ and define our Fock basis in Netket to avoid crashing the code (or maybe it's a problem just in my computer). Since Netket uses dressing libraries such as Jax to enhance computational speed, we invoke all the necessary libraries and implement a FFNN closely following *Vicentini et al*[7]. as given in the Appendix code: 1 and 2. I then use DMRG method to find the exact ground state with which I compare the results of the FFNN. See code: 3 for the implementation details of DMRG using TenPy.

Below are the convergence plots of the FFNN for different interaction strengths.



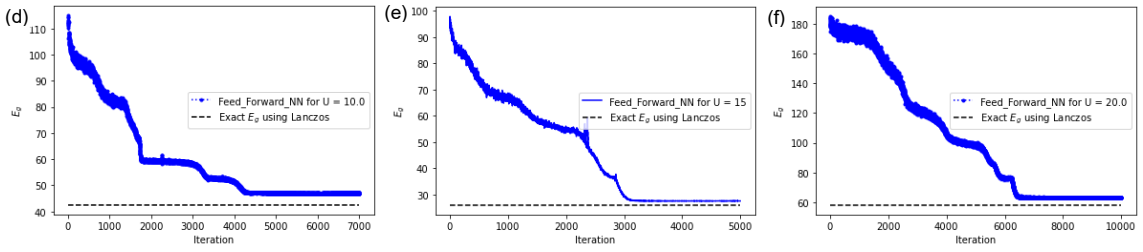


Fig. 10 Convergence of the energy value of our FFNN to the exact value predicted by Lanczos method. (a) shows a good and stable convergence. However (b) and (c) have some significant error converging to the ground state. This can happen because the the Monte Carlo sampling might be too small to represent that ground state, or we have lots of degeneracies with very little energy deviations that make it difficult to converge to a minimum with ease. (d), (e) and (f) converge to a value with good precision after sufficient amount of training iterations.

While ensuring convergence, it is important to note two things. There might be instances where the learning rate γ must be adjusted so that the convergence does not have too much divergence or errors. Another issue inherent with our Hamiltonian is that convergence can be troublesome at regimes where U is very large. This is because large diagonal values too much dominate changes in the Hamiltonian. We might get large spikes in energy change if our system changes its configuration corresponding to the diagonal entries. So in those cases, we implement a slight diagonal shift in our Hamiltonian and then optimize the altered Hamiltonian. Such a process is seen to stabilize the convergence, as shown in Fig. 11. However, it comes at the cost of the accuracy of our ground state to the actual one since it is now optimizing a slightly different Hamiltonian. This is seen in Fig. 10 (d), (e), and (f).

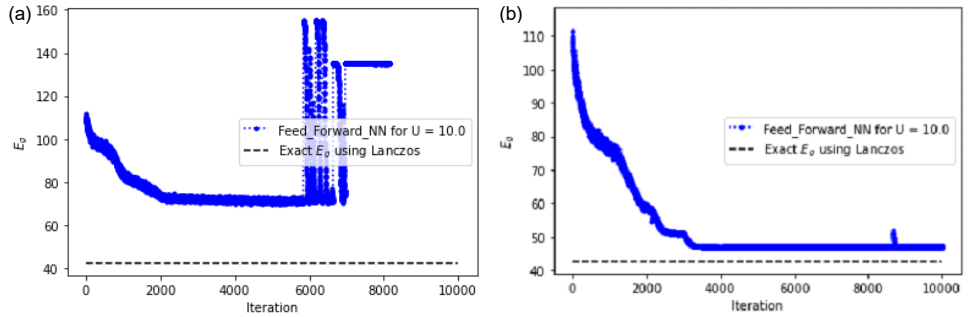


Fig. 11 (a) Convergence of our FFNN for diag_shift = 0.35. (b) Convergence of our FFNN for diag_shift = 2.0

The ground state energy predicted for various U/J values are plotted below. We see that the FFNN manages to predict the ground states with some errors while still retaining the required signatures.

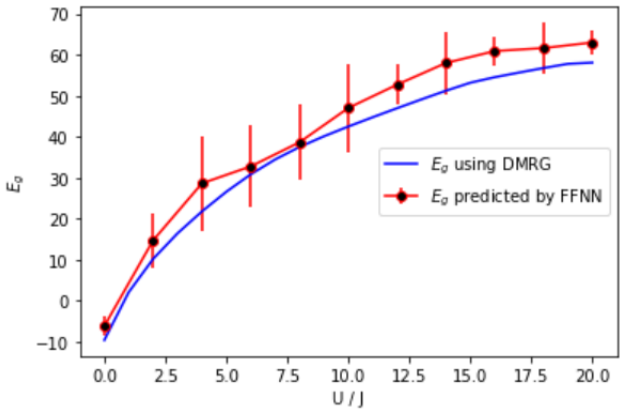


Fig. 12 Comparison of the ground states obtained by FFNN and DMRG for various interaction strength U/J

Now I shall go to the main plots, where we will see the phase transition happening in our lattice. The occupation plot of our ground state for the defined BHM is given below. Even though the FFNN had errors in converging to the proper ground state energy without significant error, it still managed to capture the phase transition from a superfluid to a Mott insulator. The FFNN has an accurate occupation for more considerable interaction strengths, although it shows slight errors for lower values of U/J. This might be because we excluded bosonic states with more than seven bosons in a site. The states close to the superfluid regime have a significant contribution from these states, which might be why our FFNN incorrectly predicts their particle occupation at each site.

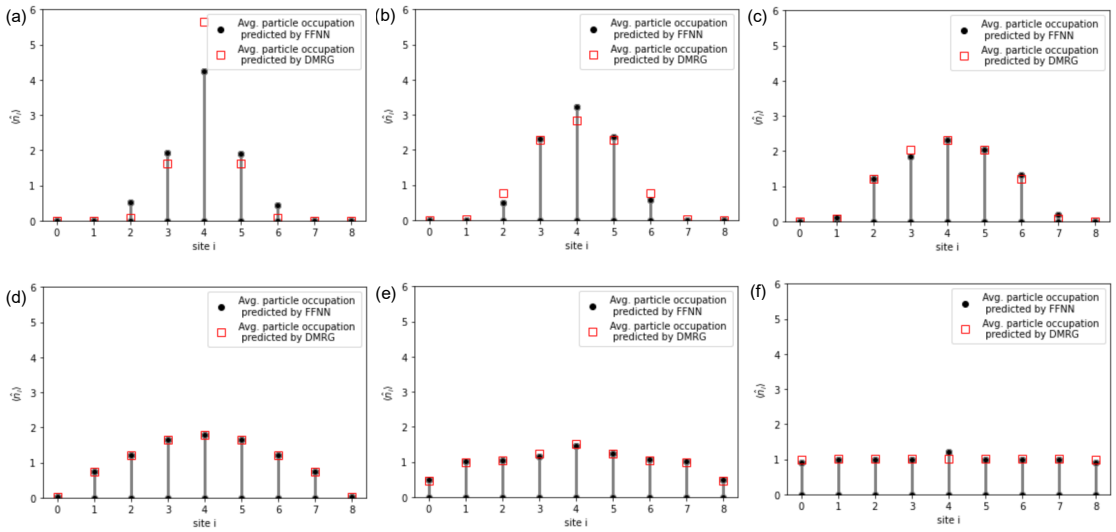


Fig. 13 Average particle occupation for various sites. (a) shows the ground state for $U/J = 0$, (b) $U/J = 2$, (c) $U/J = 4$, (d) $U/J = 8$, (e) $U/J = 16$, (f) $U/J = 20$.

The system has a binomial distribution for the superfluid regime at $U/J = 0$, where all particles exist in the $k = 0$ state. They all exist with equal probability on a position basis, as predicted in Eq. 15. As we increase the interaction strength, we see the particles slowly occupying the edge sites. For $U/J = 20$, we have a single particle or integer occupation at every site. This is the phase of a Mott insulator as described in Eq. 8, and as we have also seen that the place where our system has integer filling, we have a band gap in the Mean field diagram, we also conclude that our system is in a Mott insulating state.

Conclusion

I have explained the physics of BHM and showed how Mott insulation arises from superfluidity for a collection of bosons in a 1D lattice. I have also described machine learning concepts and shown how they can be used to solve our quantum many-body physics. I have also successfully created a FFNN model that efficiently stores information on the ground state probability. This model can spit out each coefficient of the Fock space corresponding to the ground state coefficient. This machine has successfully learned our Hamiltonian with just (9*20+20) + (20*2+2)=242 variables! A significant improvement in the computational space compared to the monstrous 24,310 states we have to define to exactly solve the Hamiltonian. In general, the space complexity for exact diagonalization is O(M!N!), whereas, for a FFNN, it is just O(M Subscript[N, H])*# iterations which is a polynomial complexity in the end! Hence, machine learning is an excellent candidate to address the issue of running out of space due to the exponentially large Fock states. Although the FFNN gives some errors in converging to the ground state, it still captures the information about the phase transition happening in our system.

Acknowledgments

I thank my senior, Vaishaki di for sharing resources on Netket-3 with which I was able to construct the neural networks, and also Rajashri Parida for sharing notes on the Bose-Hubbard model in the context of superfluid to Mott insulator phase transition. I thank Prof. Anamitra Mukherjee for encouraging me to take this interesting project and introduce me to a whole new branch of many-body physics with inter-disciplinary aspects. I thank International Centre for Theoretical Physics (ICTP), Trieste for providing Youtube lectures on Machine learning and its applications in many body physics. I also thank the Github group of Netket-3 led by Filippo Vicentini for addressing all my issues with implementing and debugging my network. Their prompt responses to my issues have greatly helped me finish this project with ease. And lastly, I thank myself for achieving the feat of finishing this project all by myself in just a month's time (gosh, I'm so proud)!

Appendix

Here I provide the main codes I have used to obtain the ground state of the defined Hamiltonian. I have not included importing, exporting, and plotting codes.

Hilbert space and the Hamiltonian:

```
import os
os.environ["JAX_PLATFORM_NAME"] = "cpu"

# Import netket library
import netket as nk
print('Version of Netket: ',nk.__version__)

# Import Json, this will be needed to load log files
import json

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
import time
import flax.linen
import jax
import jax.numpy as jnp

Lambda = 6  # cutoff of each bosonic Fock space
N = 9  # number of bosons in our system
L = 9 # no of sites in our bosonic system
g = nk.graph.Hypercube(length=L, n_dim=1, pbc=True)

hi = nk.hilbert.Fock(n_max = Lambda, N = L, n_particles = N)  # n_max -> Maximum occupation for a
site

#the system parameters

V = 1.0
J = 1.0
U = 5.0

from netket.operator.boson import create, destroy, number

# we loop over all "sites" -> loop over N bosons. PBC included

H_hop = sum([-J * (create(hi, i) * destroy(hi, (i+1)%L) + create(hi, (i+1)%L) * destroy(hi, i))
for i in range(L)])

H_onsite = sum([V * (i - L//2)**2 * (create(hi, i) * destroy(hi, i)) for i in range(L)])

H_interact = sum([U/2 * number(hi, i)*(number(hi, i) - 1) for i in range(L)])

H = H_hop + H_onsite + H_interact

eig_vals = nk.exact.lanczos_ed(
  H, compute_eigenvectors=False
)

print(eig_vals[0])
```

Code: 1 Defining the Hilbert space and the Hamiltonian whose the ground state we shall find

FFNN and Monte Carlo iterations:

```
# defining a feed forward NN with 20 hidden features
class Model(nk.nn.Module):
```

```
    @nk.nn.compact
    def __call__(self, x):

        W0 = nk.nn.Dense(features=20, dtype=np.complex128,
kernel_init=jax.nn.initializers.normal(stddev=0.1),
bias_init=jax.nn.initializers.normal(stddev=0.1))(x)

        W1 = nk.nn.Dense(features=2, dtype=np.complex128,
kernel_init=jax.nn.initializers.normal(stddev=0.1),
bias_init=jax.nn.initializers.normal(stddev=0.1))(W0)

        y = nk.nn.activation.tanh(W1)
        return jax.numpy.sum(y, axis=-1)

ffnn = Model()

# Create the a metropolis exchange algo to predict the probability of going bw two states
sampler = nk.sampler.MetropolisExchange(hi, graph = g)

# create the VMC state
vstate = nk.vqs.MCState(sampler, ffnn, n_samples = 1024)

optimizer = nk.optimizer.Sgd(learning_rate = 0.0003)
preconditioner = nk.optimizer.SR(diag_shift = .31)

# Notice the use, again of Stochastic Reconfiguration, which considerably improves the optimisation
gs = nk.driver.VMC(
    H, optimizer, variational_state = vstate, preconditioner = preconditioner
)

iter = 3000

# logging and running : sometimes need to rung longer than 300 to exit a plateaux
log = nk.logging.RuntimeLog()
gs.run(n_iter = iter, out = log)

ffn_energy = vstate.expect(H)
error = abs((ffn_energy.mean - eig_vals[0]) / eig_vals[0])
print(f"Optimized energy: {ffn_energy} \nRelative error: {error*100:.2f}%")
```

Code: 2 Code to create a FFNN and train it using variational Monte Carlo method

DMRG to compare it with FFNN results:

```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

# library to define our BHM
from tenpy.models.hubbard import BoseHubbardModel, BoseHubbardChain

# library to run our DMRG algo
from tenpy.networks.mps import MPS
from tenpy.algorithms import dmrg

# Model parameters
N = 9  # number of bosons in our system
L = 9 # no of sites in our bosonic system
# implement finite DMRG on our BHM
bc = 'finite'

V = 1.0
J = 1.0
```

```
U = 20.0
mu = 0

v_arr = np.array([V*(i-L//2)**2 for i in range(L)])

model_params = {
    'bc_MPS': bc, #to assert finite lattice DMRG
    'bc_x': 'periodic', # considering PBC
    'conserve': 'N', # hamiltonian conserves particle number
    'L': L, # length of our finite lattice
    't': J, # hopping strength
    'V': 0, # nearest neighbour interaction NOT on-site potential term
    'U': U, # on-site interaction
    'mu': -v_arr + mu, # on-site energy term
    'lattice': 'Chain', # asserting our lattice is a 1d chain
    'n_max': L # max. allowed of bosons in each site
}

model = BoseHubbardModel(model_params)

# we input an initial config with one boson on each site for the algo to process and find the gs
init_config = [1]*L

print('initial configuration as a vector: ',init_config)

#convert the vector to a MPS
psi = MPS.from_product_state(model.lat.mps_sites(), init_config, bc=bc)

print('Matrix product state defined: ', psi)

# make the dmrg engine to implement optimization of our model
dmrg_params = {
    'mixer': None, # setting this to True helps to escape local minima
    'max_E_err': 1.e-10, # the error tolerance for our DMRG iteration
    'trunc_params': {
        'chi_max': 100, # max bond dimension
        'svd_min': 1.e-10, # min svd value that is accepted
    }
}
eng = dmrg.TwoSiteDMRGEngine(psi, model, dmrg_params)
E, psi_ground = eng.run() # the main work; modifies psi in place

# the ground state energy was directly returned by dmrg.run()
print("ground state energy = ", E)

# there are other ways to extract the energy from psi:
E1 = model.H_MPO.expectation_value(psi_ground)  # based on the MPO
print(E1)
```

Code: 3 DMRG implementation of the BHM and solve for its ground state

References

A. [1] Philipp M. Preiss and Ruichao Ma and M. Eric Tai and Alexander Lukin and Matthew Rispoli and Philip Zupancic and Yoav Lahini and Rajibul Islam and Markus Greiner. Strongly correlated quantum walks in optical lattices. Science (2015). https://doi.org/10.1126/science.1260364

B. [2] Paolo Molignini. Analyzing the two dimensional Ising model with conformal field theory. Departement Physik, ETH Zurich (2013)

C. [3] https://github.com/TibDoicin/Mean-Field-Theory-Bose-Hubbard-Model

D. [4] S. Ejima1, H. Fehske and F. Gebhard. Dynamic properties of the one-dimensional Bose–Hubbard model. EPL(2011)

E. [5] Giuseppe Carleo a,∗, Kenny Choo b, Damian Hofmann c, James E.T. Smith d, Tom Westerhout e, Fabien Alet f, Emily J. Davis g, Stavros Efthymiou h, Ivan Glasser h, Sheng-Hsuan Lin i, Marta Mauri a,j, Guglielmo Mazzola k, Christian B. Mendl l, Evert van Nieuwenburgm, Ossian O'Reilly n, Hugo Théveniaut f, Giacomo Torlai a,
Filippo Vicentini o, Alexander Wietek. NetKet: A machine learning toolkit for many-body quantum systems. SoftwareX 10 (2019) 100311. https://doi.org/10.1016/j.softx.2019.100311

F. [6] Kristopher McBrian et al 2019 J. Phys.: Conf. Ser. 1290 012005.

G. [7] Filippo Vicentini, Damian Hofmann, Attila Szabo, Dian Wu, Christopher Roth, Clemens Giuliani, Gabriel Pescia, Jannes Nys, Vladimir Vargas-Calderon, Nikita Astrakhantse and Giuseppe Carleo. NetKet 3: Machine Learning Toolbox for Many-Body Quantum Systems. SciPost Physics Codebases (2021)