Discovering Phases of Many-Body Systems with Unsupervised Learning

Benjamin Safvati
Stanford University
Physics 212 Final Project

November 11, 2022

1 Introduction: Finding Efficient Descriptions of Many-Body Physics

---

The emergent properties of interacting particles are of great interest in condensed matter physics for the engineering of exotic quantum materials [12] and understanding of phenomena unexplained by the symmetry breaking paradigm in Ginzburg-Landau theory. Novel phases of matter, like those seen in the fractional quantum hall effect [14] and high-temperature superconductors [5], are indiscernible from the physics of individual particles, motivating new approaches to accurately simulate the quantum behavior of bulk ensembles of particles with strong interactions. The accessible states of many-body quantum systems notoriously live in a Hilbert space exponentially large with respect to system size, and so accessing a complete description of the many-body wavefunction is found to be unattainable with classical simulation techniques.

Modern approaches to the many-body problem have found success by employing efficient variational models that represent the wave function with polynomial scaling in system size. Tensor Network States, which represent the wavefunction by the product of variational tensor elements, excel at representing the ground states of realistic Hamiltonians in nature, where interactions between particles are local and the entanglement entropy of a sub-region scales with the sub-system boundary, rather than the volume of the region [8]. This constraint exponentially reduces the space of relevant states and motivates the design of Tensor Network States with varied entanglement structures for different systems, including Matrix Product States (MPS) for local lattice Hamiltonians in 1D and the Multiscale Entanglement Renormalization Ansatz (MERA) for systems near criticality [17]. A more recent technique developed by Carleo and Troyer [2] uses a Neural Network representation of the wavefunction called a Neural Quantum State (NQS), where the complex amplitudes in a configuration basis are found as a function of the configuration and a set of hidden "neurons" that connect between sites and learn the relevant particle correlations. This machine learning-inspired ansatz was then used for a feedback-based stochastic optimization algorithm that accurately found the ground state of various local spin-$1/2$ Hamiltonians. Other work has made explicit transformations between the two variational models [3, 11], showing how a NQS with only local connections displays area-law entanglement and can be mapped to a MPS, whereas a Neural Network with dense connections has maximal volume-law entanglement scaling and no efficient Tensor Network structure. [6]

The variational methods above rely on an efficient choice of wavefunction Ansatz, which can be difficult to devise without prior knowledge of the many-body Hamiltonian. Furthermore, such algorithms are mostly focused on simulation of the system ground state, ignorant of the emergent phases that appear when temperature or other system parameters are varied. While conventional phase transitions can be identified by the labelling of sample data with the corresponding local order parameter (e.g. magnetization), new phases of matter without obvious local identifiers are considerably harder to identify. Topological phase transitions in particular are difficult to simulate because the identifiers of such transitions are non-local in the sense that they are invariant to continuous deformations of the lattice. [1] While more advanced model architectures can be designed to account for topological features [15, 7], these methods still require labelling of training data with the appropriate topological invariant [23] and thus cannot be used to discover unknown phases.

In addition to the generation of statistical models, there exist machine learning algorithms that learn efficient representations of the state space directly from raw data. Such dimensionality reduction methods, called unsupervised learning, elucidate the relevant features for classifying different sample configurations without any a priori knowledge of where the samples came from, and so they can be broadly applied to the analysis of many-body systems with unknown phase behavior and arbitrary interactions. This paper will demonstrate the effectiveness of different lower dimensional data reconstruction algorithms for identifying both symmetry-breaking and topological phases directly from Monte Carlo sampled data. The failure of standard reconstruction algorithms to identify topological order is linked to the notion of similarity used to compare samples in the dimensionality reduction process. Notably, Euclidean measures of distance such as the L2 norm are unsuited for the characterization of topological phases [1], suggesting a global connectivity in the state space geometry of such systems that is not present for symmetry-breaking phases.

Using the 2D Ising Model as an introduction to the techniques, we will show how simple analysis of the data covariance matrix can distinguish disordered and ordered phases of the system and discover the magnetization as the appropriate local order parameter of the transition [22]. By projecting data onto the axes of the learned features, one can use clustering algorithms to define boundaries between subsets of samples in different phases and thus approximate the critical temperature of the transition. For this project, the sample data from the 2D Ising model will be generated by a Neural Quantum State architecture [2], whose training data is uniquely self-generated by a stochastic optimization procedure that we have implemented. As the NQS algorithm iteratively approaches the ground state, an effective temperature of the model can be related to the system temperature to diagnose the crossing of a critical point.

Next, we will show how conventional methods fail to classify topological phases in the 2D XY model, where the Kosterlitz–Thouless (KT) transition describes the binding of free vortex-antivortex pairs as the temperature is decreased. To overcome the obstacle of accounting for global structure in the samples, we utilize a nonlinear dimensionality reduction method that defines similarity between two configurations by a diffusion process across the sample space. In this formulation a random walk is defined between samples and global features are captured by the path distance after many time steps of the walk process [4]. This method is able to resolve different topological phases in the XY model at low temperatures and visualize the proliferation of vortices by the breakdown of topological order as the temperature crosses the KT transition point. The ability to classify topological phases from randomly generated configurations of the XY model makes unsupervised learning algorithms useful tools for the characterization of exotic phases of matter directly from Monte Carlo data.

2 Learning Phases of the 2D Ising Model

2.1 Principal Component Analysis

The great advantage of unsupervised learning techniques is that they discover efficient representations of data directly from a sample set without the need for a complex variational model with many parameters to optimize. As big data becomes more prevalent throughout different fields in science and technology, unsupervised learning has been extensively used as a guide for determining what features best capture information about the data space. These features are typically built as linear combinations of the data variables; for this paper we restrict our attention to systems of N particles on an $L \times L$ 2D square lattice where each data vector $x = (x_1, x_2, ..., x_N)$ is N-dimensional and each data variable x_i equals the value of the particle degree of freedom at site i. The learned features of statistical physics models should then be able to capture the same information as local order parameters like the magnetization that also depend on linear combinations of particle values.

The characteristic algorithm in unsupervised learning, known as Principal Component Analysis (PCA), has existed for more than a century [18] and is deeply rooted to basic ideas in Linear Algebra. The construction involves only an $m \times n$ data matrix X where each row is a data point in the n-dimensional data space and each column is normalized to have zero variance. Then the principal components are defined as the eigenvalues of the covariance matrix $X^\top X$ with the corresponding eigenvectors describing the directions of maximum variation in the data. Then by projecting the data onto the basis defined by the first few principal directions, one can visualize the data in a lower dimensional space with minimal information loss with respect to the expectation of the sample variability. More formally, the task of PCA can be defined as finding a vector w^* such that

$$w^* = \max_{w \in \mathbb{R}^n} w^\top X^\top X w \quad \text{s.t } w^\top w = 1$$

Then from simple knowledge of linear algebra we know that w^* is the eigenvector of $X^\top X$ with maximum eigenvalue, and this exactly corresponds to the feature with maximum variance with respect to X. The next principal component can similarly be found by maximizing the variance with an added constraint that the second weight vector must be orthogonal to the first, creating a basis of principal vectors in a Gram-Schmidt like process. The generality of this optimization problem is no coincidence, as this same optimization becomes useful in classical mechanics, signal processing, and general data reconstruction.

2.2 Variational Optimization Methods

Because PCA for dimensionality reduction is a fairly simple technique, I have decided to demonstrate how the method can unravel characteristics of the phase transition in the 2D Ising model in comparison to a simple variational neural network model inspired by statistical physics that iteratively approaches the system ground state. It will become clear that PCA is far better suited to phase classification in comparison to variational methods that only capture the low-temperature physics of the system.

The starting point of any ground state numerical optimization algorithm is the variational principle in Quantum Mechanics, which allows us to formulate the energy expectation as the objective function for finding the ground state of the Hamiltonian with the minimization problem

$$|\Psi_0\rangle = \min_{|\Psi\rangle} \langle\Psi| H |\Psi\rangle .$$

Although this quadratic form still suffers from exponentially large dimensions relative to system size, a landmark result by McMillan [16] shows how this expectation can be calculated numerically by averaging over the probability distribution of the target quantum state. We show this by algebraic manipulation of the objective function, noting that we can rewrite the quadratic form as a sum over pairs of configurations (S, S') where the element $H_{S,S'} = \langle S| H |S'\rangle$ is explicitly operated on in the calculation below

$$\frac{\langle\Psi| H |\Psi\rangle}{\langle\Psi| |\Psi\rangle} = \frac{\sum_{S,S'} \psi(S)^* \langle S| H |S'\rangle \psi(S')}{\sum_S |\psi(S)|^2}$$

$$\frac{\langle\Psi| H |\Psi\rangle}{\langle\Psi| |\Psi\rangle} = \frac{\sum_S |\psi(S)|^2 (\sum_{S'} H_{S,S'} \frac{\psi(S')}{\psi(S)})}{\sum_S |\psi(S)|^2}.$$

In this form the expectation value of the energy can be interpreted as a statistical average over the probability distribution determined by the quantum state. The quantity $E_{loc}(S) = \sum_{S'} H_{S,S'} \frac{\psi(S')}{\psi(S)}$, known as the local energy of configuration S, is efficiently computed for local Hamiltonians where the sparsity of non-zero elements is guaranteed. For the 2D Ising Model there are no off-diagonal terms because the Hamiltonian is diagonalized in the single-particle basis, and so the local energy reduces to the energy of a given configuration.

Because neural networks often settle at local minima near the global minimum in the parameter space, it is instructive to estimate the energy variance at each iteration of the training. Because the ground state is an eigenstate of the Hamiltonian, the energy variance should be exactly zero:

$$\sigma_H^2 = \langle H^2\rangle - \langle H\rangle^2 = \langle\psi_0| H^2 |\psi_0\rangle - (\langle\psi_0| H |\psi_0\rangle)^2$$

$$\sigma_H^2 = E_0^2 \langle\psi_0| |\psi_0\rangle - E_0^2 = 0.$$

From the energy objective function the direction of steepest descent towards the ground state can be found by taking the gradient with respect to each variational parameter in the quantum state model, allowing us to update the parameters iteratively. Shaping the model parameters into a vector $\theta \in \mathbb{C}^{N_{var}}$ where N_{var} is the number of model parameters, θ is updated according to the following rule

$$\theta^{t+1} = \theta^t - \gamma_t F^t$$

where F is the gradient of the energy function, referred to as the generalized forces of the system in analogy with Lagrangian Mechanics. The learning rate γ scales the update, and by slowly decreasing this value the temperature of the system can be gradually brought down and the system can settle to the global minimum, a technique known as simulated annealing in optimization literature. Define the variational derivative of the ith parameter as

$$O_i(S) = \frac{1}{\psi(S)} \frac{\partial}{\partial \theta_i} \psi(S)$$

(this is the energy gradient for a model resembling the Boltzmann distribution, which will be true for the neural network in this project). Then the generalized force for the ith parameter can be framed as an expectation over the quantum state probability distribution (see [2] for a derivation), where

$$F_i = \langle E_{loc} O_i^* \rangle - \langle O_i^* \rangle \langle E_{loc} \rangle.$$

This gradient expression iteratively updates the parameters by finding an approximation of the energy gradient based on a random sampling of the full data set. This process is referred to as stochastic gradient descent because the full gradient is never explicitly solved, and the updates calculated have some variance due to the finite sampling. The accuracy of the gradient descent method depends on the set of sample configurations $\{S_i\}$ used to compute the expectations involved in the stochastic parameter updates. Because the model approximates the ground state $|\Psi_0\rangle$, we want the expectation to be over the complex probability distribution determined by the amplitudes of $|\Psi_0\rangle$. Starting with randomly assigned parameters, this distribution can be approached asymptotically with the Metropolis-Hastings algorithm. To create samples, random walks in the space of spin configurations are taken and the new configurations are accepted with probability determined by the ratio of quantum state amplitudes. This process has a Markov chain interpretation [2] where a new spin configuration is added to the chain with probability

$$P(S^{k+1}) = \min\left(1, \left|\frac{\psi(S^{k+1})}{\psi(S^k)}\right|^2\right).$$

After several thousand iterations we have a large enough training set to compute parameter updates using Stochastic Gradient Descent. Samples should be acquired after every some amount of Metropolis steps, allowing proper time for the Markov Chain to mix so that the samples are uncorrelated. The theory requires that the samples should be accepted from the chain each time the autocorrelation time has passed, but in practice taking a number of steps equal to the number of variational parameters will suffice for proper sample mixing.

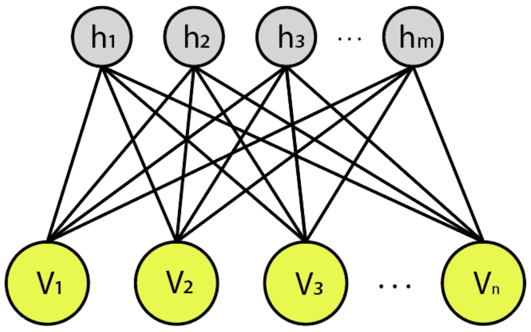


Figure 1: A Restricted Boltzmann Machine (RBM) with a visible layer representing the spin degrees of freedom at each site on the spin chain, as well as a hidden layer of neurons that keep track of correlations between spins. As a generative model, the RBM models a joint distribution over all of the network units, and the wavefunction amplitudes can then be found by tracing out the hidden units.

The choice of neural network architecture is highly non-generic and different models have been shown to faithfully represent the many-body wavefunction for various kinds of systems [2, 20]. For our purposes a practical choice is the Restricted Boltzmann Machine (RBM) [9], an energy-based model that resembles the canonical ensemble in statistical mechanics. This network consists of a layer of n visible neurons $\sigma_1, ..., \sigma_n$ that correspond to the spin degrees of freedom attached to a layer of m hidden units $h_1, ..., h_m$ that model the relevant correlations in the spin chain. Then the energy of a given configuration is found as

$$E(\sigma, h; \theta) = -\sum_{j=1}^{n} a_j \sigma_j - \sum_{i=1}^{m} b_i h_i - \sum_{i=1,\, j=1}^{m,\, n} W_{ij} h_i \sigma_j$$

where $\theta = (a_j, b_i, W_{ij})$ are the parameters of the neural network. A quantum state can then be expressed as the Boltzmann weight for the energy of that configuration,

$$\psi(\sigma) = \sum_{\{h_i\}} e^{-E(\sigma,h;\theta)} = \sum_{\{h_i\}} \exp\left(\sum_{j=1}^{n} a_j \sigma_j + \sum_{i=1}^{m} b_i h_i + \sum_{i=1,\, j=1}^{m,\, n} W_{ij} h_i \sigma_j \right).$$

The parameters are chosen to be complex, allowing us to compute both a magnitude and phase when modelling the ground state. An important value is $\alpha \equiv m/n$, the ratio of hidden units to visible units, which characterizes the density of connections between the visible and hidden layer. In principle, by increasing α the representative ability of the RBM can be systematically improved, but the added complexity may also lead to overfitting of the model to the current training set. This ruins the generalizability of the network and leads to inaccurate estimates of observables [9].

2.3 Results for the 2D Ising Model

Now that the sampling of states and optimization procedure have been outlined, we can demonstrate how both unsupervised learning and stochastic variational optimization can be utilized together to understand the behavior of the Ising Model at different temperatures. Starting with the neural network model, we consider the 2D Ising model on a square lattice with nearest neighbor interactions, periodic boundary conditions, and no external field

$$H = -\sum_{<ij>} \sigma_i \sigma_j$$

where $\sigma_i \in \{-1, 1\}$. This system is known to exhibit a $\mathbb{Z}_2$ symmetry-breaking phase transition at $T_c \approx 2.269$, with a high-temperature disordered phase that aligns in one of two directions as a ferromagnet upon cooling of the system below the critical temperature. A system of $N = 100$ spins is modeled by a RBM with N input units and αN hidden units that iteratively generates $M = 100$ uncorrelated samples at each training step. These sample sets are stored to compute gradients of the energy in parameter space that are used to update the model parameters at each time step. The convergence of the energy over 100 training steps is good indication that the ferromagnetic ground state has been reached, and the corresponding plot of the energy variance reaching zero verifies that the trained network is simulating the ground state.

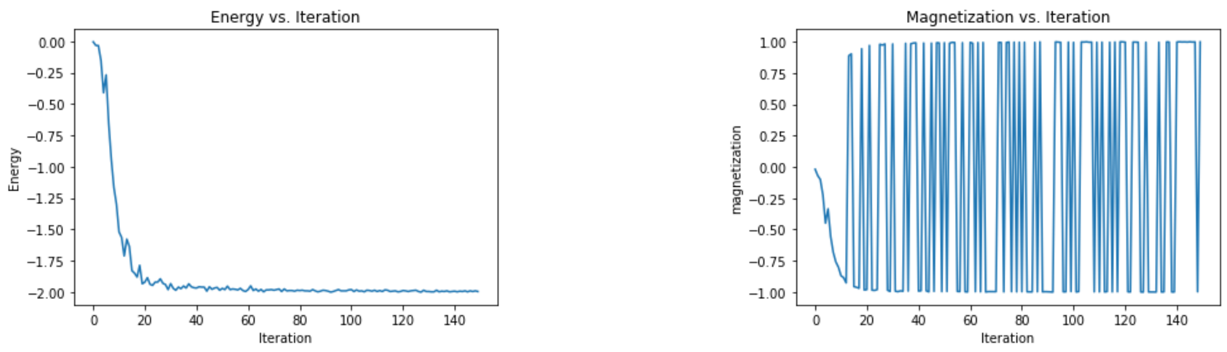
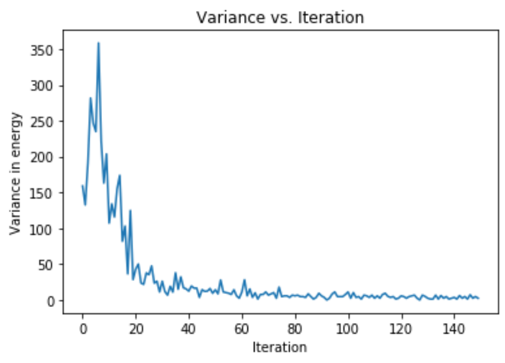


Figure 2: The estimated energy and magnetization of the 2D Ising model on a 10×10 square lattice with periodic boundary conditions. The convergence of the model after about 5 iterations signifies that the complexity of the model is sufficient for capturing the correlations between spins on the lattice. Furthermore, the magnetization oscillating between both ferromagnetic phases shows that the model is correctly traversing the full state space at each iteration and learning both ground state configurations.

The optimal hyperparameters for the model were chosen to acquire a smooth range of system configurations from the initial disordered samples to the final ferromagnetic ones. The hidden unit density α is set to .25 because a more connected model will converge to the ordered phase too quickly. At first glance, it is not clear how to obtain phase information from samples generated by this network, where no temperature is explicitly defined. But looking at the convergence of the energy it is clear that at some point the model crosses a transition, signified by a large drop in the energy and a spike in the energy variance. This behavior is present in simulations on a wide variety of lattice sizes and other tuning of hyperparameters, hinting that it may describe a universal property of the system that we are trying to simulate.



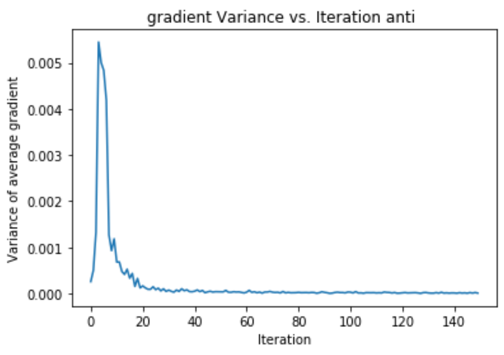


Figure 3: The estimate variance in energy (left) and in the gradient updates (right) as predicted by the model at each iteration of the stochastic optimization. The peaks of both plots at iteration 4 suggest a phase transition where the model exhibits divergent correlations and thus the greatest fluctuations in the energy as the model approaches the ordered phase.

To try and identify the point at which the model approaches the transition, we define an "effective temperature" [2] for the model at each time step that relates to the expectation of the gradient updates of all variational parameters at each time. Intuitively this gradient variance describes fluctuations in the solution space that similarly peak at the critical temperature then converge to zero in the ground state. Thus the magnitude of the gradient update determines the temperature of the model at each iteration, following the formula

$$T_{eff} = \frac{\gamma\langle \mathrm{Var}(\theta_i)\rangle}{2}$$

Then the critical point can be determined by evaluating the gradient variance expectation at the iteration where this transition is crossed, which for the simulation presented on 100 spins occurs at iteration 4. This point translates to an effective temperature of 2.244, an error of only about 1 percent in remarkable agreement with theory. This result is however highly dependent on the choice of model hyperparameters, where in most cases the critical phase is crossed during the thermalization steps of the Markov Chain Monte Carlo and thus never captured by the recorded data. Thus we now move to PCA as an alternative method for discovering phase transitions from the sample data generated by the neural network optimization procedure.

For the following analysis we define a new 12×12 lattice of Ising spins and choose a low hidden unit density $\alpha = .1$ so that the model does not converge until approximately iteration 20. Creating 50 uncorrelated samples every step for 100 time steps, this model produces a smooth sampling of the state space of the Ising Model across a range of temperatures in both the disordered and ordered phases. By using PCA to project the spin data onto the two most significant principal components, one can readily identify the symmetry-breaking nature of the phase transition. The magnitude of the principal components (i.e eigenvalues of the covariance matrix) reveal the relative significance of the corresponding features, and for the 2D Ising model the first principal component is by far the most informative.

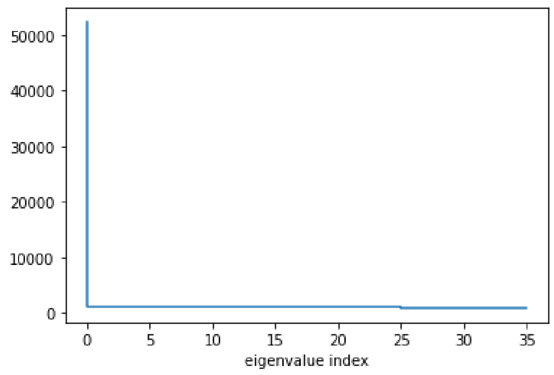


Figure 4: The spectrum of principal components (or eigenvalues of the covariance data matrix) in ascending order for a dataset gathered from a 12×12 square lattice Ising model. The extreme dominance of the first component tells us that the phase transition can be well described by a single feature, in this case the magnetization.

This vector very closely resembles the uniform vector over all sites $w_1 = \frac{1}{N}[1, 1, ..., 1, 1]^\top$, and so the projection of the spin variables onto w_1 directly resembles the measurement of the system magnetization

$$\langle \sigma \rangle = \frac{1}{N} \sum_{i=1}^{N} \sigma_i = \langle \sigma \cdot w_1 \rangle$$

Without any prior knowledge of the physical system that created the data set, PCA is able to identify the correct local order parameter that distinguishes the ordered and disordered phases of the Ising model. One can discern the bifurcation that occurs as the effective temperature is lowered by the localization of the samples on either end of the first principal axis over several iterations. The second principal component provides information about the fluctuations in different phases: the low temperature points lie exactly on the first principal axis, implying zero variance in the magnetization, whereas the disordered phase samples spread radially about the origin. The samples with the greatest spread on the 2D projection represent the system at criticality where fluctuations are largest.

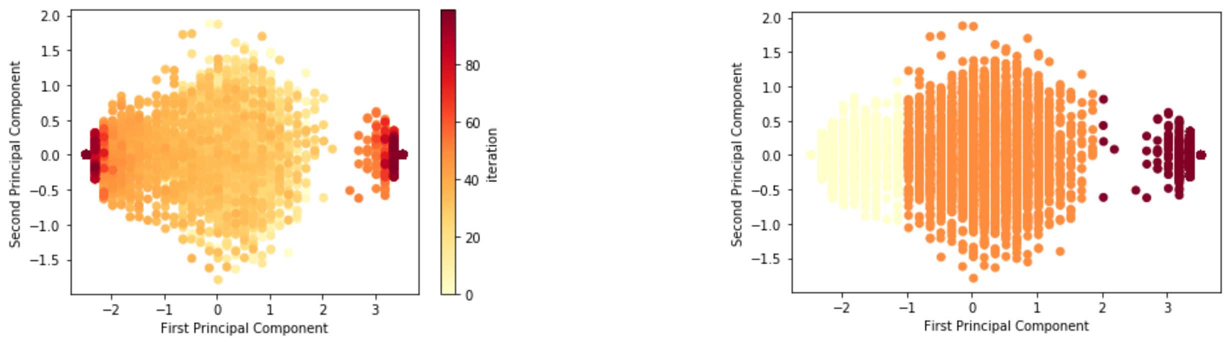


Figure 5: (left) Projection of the Ising model data onto the first two principal components. The iteration number is equivalent to an effective temperature and allows us to understand how PCA classifies the data. The first axis clearly defines the magnetization as can be seen by the fact that samples at lower temperatures acquire a magnetization along one of two directions with equal magnitude. (right) the result of K-means Clustering of the projected data, which identified correctly the disordered phase at the center of the plot and the two ferromagnetic phases at the ends.

This lower dimensional representation of the data can be very easily used to determine the critical temperature using clustering algorithms that seek to partition the data into different classes. The most common algorithm, known as K-means clustering, seeks to maximize the distance between each cluster as well as the inter-cluster distance to each cluster's center, and when applied to this dataset immediately acquires the phase distribution as a function of the magnetization. With data generated by Monte Carlo at varied temperatures, clustering algorithms provide an alternative method to approximating the critical temperature.

3 Diffusion Maps

Principal Component Analysis has been shown to elucidate key aspects of the Ising phase transition in 2D, but for topological phase transitions without suitable local descriptions, this technique has been shown to be unsuccessful [10, 1, 19]. PCA in itself is a fairly simple method, only capable of creating features out of linear combinations of the data space, and so it is not surprising that such non-local orders would be more difficult to identify. PCA is implicitly defined on a Euclidean metric, where distance can be measured by the L2 norm. Such local notions of similarity cannot capture the kind of relationships that partition topological phases, which cannot be broken by continuous deformations and thus carry a global structure. An illustrative example is the topological order in the 2D XY model, where spins are described by unit vectors on circles with nearest neighbor interactions. Here the relevant quantity for classifying each phase is known as the winding number: this topological invariant measures the number of rotations that the spins undergo along a closed curve within the lattice. Importantly, this curve can be of any size and must only be closed, an indicator of the topological structure of such phases. The winding numbers can be related to the emergence of vortices and antivortices in the XY model, swirling excitations that condense into pairs of net winding number zero when the temperature is low and break free above a critical temperature $T/J \approx 0.89$, creating divergent vortex excitations that lead to a disordered phase.

Kernelized extensions of PCA have been devised to include nonlinearity with respect to the original data, where the principal components are never explicitly solved for and a kernel function is used to project data onto implicitly defined principal components in a nonlinear feature space [21]. A popular choice of kernel is based on a gaussian function

$$K(x_1, x_2) = \exp\left(-\frac{||x_1 - x_2||^2}{2N\varepsilon}\right)$$

where in the XY model $||x_1 - x_2||^2/2N = 1 - \frac{1}{N}\sum_{i=1}^{N} \cos(\theta_1 - \theta_2)$ [19] and the hyper-parameter ε determines the variance of the kernel, or more intuitively the bandwidth at which long-range connectivity is measured by the function. The nonlinear feature of the kernel method makes it attractive for interpreting long-range similarity in data, but these techniques are highly dependent on the choice of kernel function [19] and fail to generalize well to randomly generated data sets.

A powerful technique used in machine learning involves the generation of a graph-based model for a data set (this is the basis of PageRank). The weights between different data points allow the construction of a local geometry within the state space by defining a Markov process between samples and using the transition probability matrix to describe distance in the manifold that the data exists in. The method known as diffusion maps [4] uses this idea to define a dynamical distance function for the dataset that at long times carries information about the global connectivity of the sample space. The technique defines a random walk based on a Markov process between samples and uses the long time path behavior of this walk to interpret the manifold geometry at different length scales. Formally, one can construct the probability transition matrix P from the data such that

$$P_{i,j} = \frac{K_\varepsilon(x_i, x_j)}{z_i}$$

where z_i is a normalizing factor to make the transition matrix a well-defined probability distribution along the rows of P. This matrix defines a single-step transition probability, but to capture global features a diffusion distance is defined at time step t that uses higher powers of the transition matrix to capture information about longer walks on the Markov chain. This distance function allows us to define an embedding $x \to \Phi$ of the data onto the eigenvectors ψ_i of the transition probability matrix, with weighted distance along each component that is time dependent,

$$D_t(x_i, x_j) = ||\Phi_i - \Phi_j||^2 = \sum_{k=1}^{m-1} \lambda_k^{2t}((\psi_k)_i - (\psi_k)_j)^2$$

where λ_k are the eigenvectors of the transition probability matrix. The fact that λ_k are raised to higher powers as time goes on reflects that the diffusion process approaches the long-time path behavior of random steps within the data manifold. The projection of the features along the eigenvectors of P cluster the data by their global structure that relies on both the stationary state behavior after many time steps and the local connectivity of each sample.

4 Results for the 2D XY Model

Samples for the 2D XY model were generated for a 10×10 square lattice of spins defined on respective unit circles with nearest neighbor interaction of unit strength and periodic boundary conditions. At each step of the Monte Carlo sampling a single Metropolis step is done for each lattice site. The samples are taken and analyzed at fixed values of T/J below, above, and near the critical point of the system. The diffusion map projections inform about the fidelity of the topological order below the critical temperature, and by slowly sweeping T/J around T_c we can visualize the breakdown of topological order [19] and the proliferation of vortices creating disorder that has no lower dimensional representation. By using K-means clustering on the projected data and observing the value of T/J at which the clustering fails, we can estimate the critical point of the KT transition simply from Monte Carlo generated XY data.

$$T/J = 0.3$$

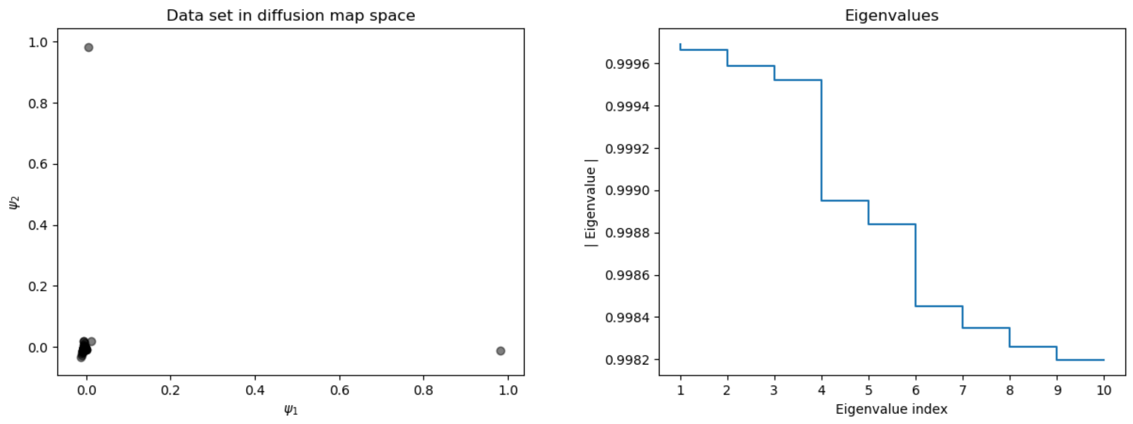


Figure 6: (left) The projection of the 2D XY model onto the two dominant eigenvectors of the transition probability matrix for $T/J = 0.3$. At low temperatures this model exhibits a topological order determined by the three clusters identified by the algorithm. These clusters naturally relate to the emergence of single vortex excitations in the x and y direction. (right) The near-threefold degeneracy of the transition probability matrix further suggests a ground state degeneracy at low temperatures that is classified by a topological invariant.

The low temperature phase of the 2D XY model is known to exhibit topological order, and the diffusion map process is able to characterize samples by their topological invariant without any knowledge of the winding number in the simulation process. In 2D the winding number is defined as (ν_x, ν_y), and the three clusters in the reconstruction represent the completely bound vortex-antivortex case for samples at the origin and single vortex low energy metastable excitations for the smaller cluster samples [13]. The degeneracy of the transition probability matrix is key to understanding the necessary projections needed to fully characterize the different topological phases, and in this case the third dimension reveals the corresponding topological clusters for anti-vortex low energy perturbations. Higher order vortex excitations ($|\nu| > 1$) are not seen, likely because the lattice side lengths we consider do not allow for stable energy configurations with higher order vortices.

$$T/J = 10$$

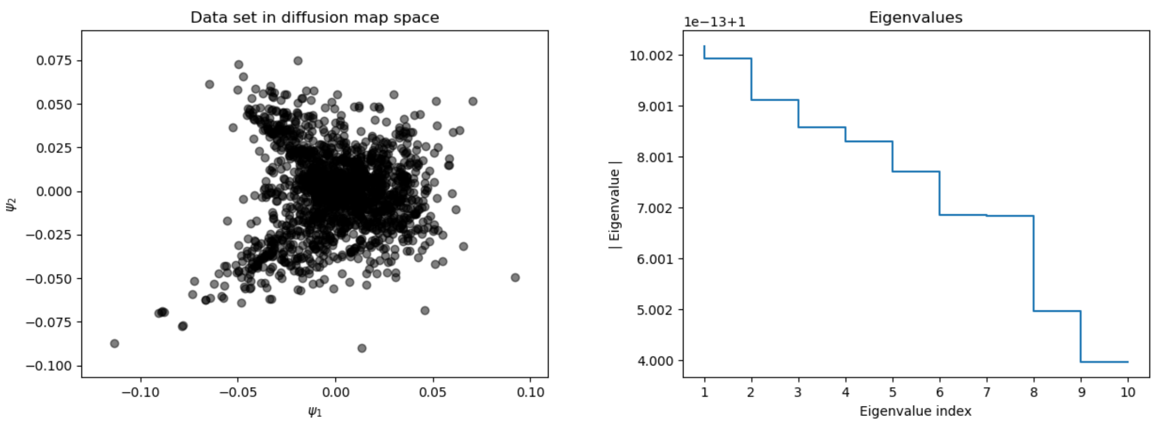


Figure 7: (left) The projection of the 2D XY model onto the two dominant eigenvectors of the transition probability matrix for $T/J = 10$. At temperatures far above the critical point no topological order remains and no efficient representation of the data can be found. This disordered phase can be understood as the result of the thermal energy reaching the binding energy of vortex-antivortex pairs, leading to free vortices throughout the lattice. (right) The spectrum in the high temperature phase shows no degeneracy and no dominant eigenvector, suggesting the diffusion process can't find a lower dimensional representation of the data sampled in this phase.

In the other limit of high temperature, the threefold degeneracy and gap in the spectrum is erased and a disordered phase emerges with no lower dimensional representation. Interested in qualitatively estimating the critical temperature, we create projections of samples taken at temperatures $T/J = 0.87, 0.89.0.91$ to see how sharply the phase transition manifests itself in the breakdown of the topological clusters. We see in the figures below that at the critical temperature the spread of samples about the origin hint at the breaking of the topological order. Above this point, we can see the data points begin to spread beyond the topological clusters, although this spread is farther smaller than the high temperature phase for small increments above T_c. To determine the critical point with reasonable accuracy, one must perform these diffusion mappings for various values of the kernel variance ε. Then one would use cluster algorithms to partition the different phases in each mapping, and the final critical point can be found as the average of the estimated critical points at each scale. For large enough sample sets this procedure should give identical critical points for a wide range of ε, but this would take too long with my limited resources. Still, this qualitative analysis proves that Diffusion Maps are powerful tools for the identification of topological order, able to sort samples into clusters by their topological invariant and diagnose the transition to an ordered phase upon crossing the KT point.

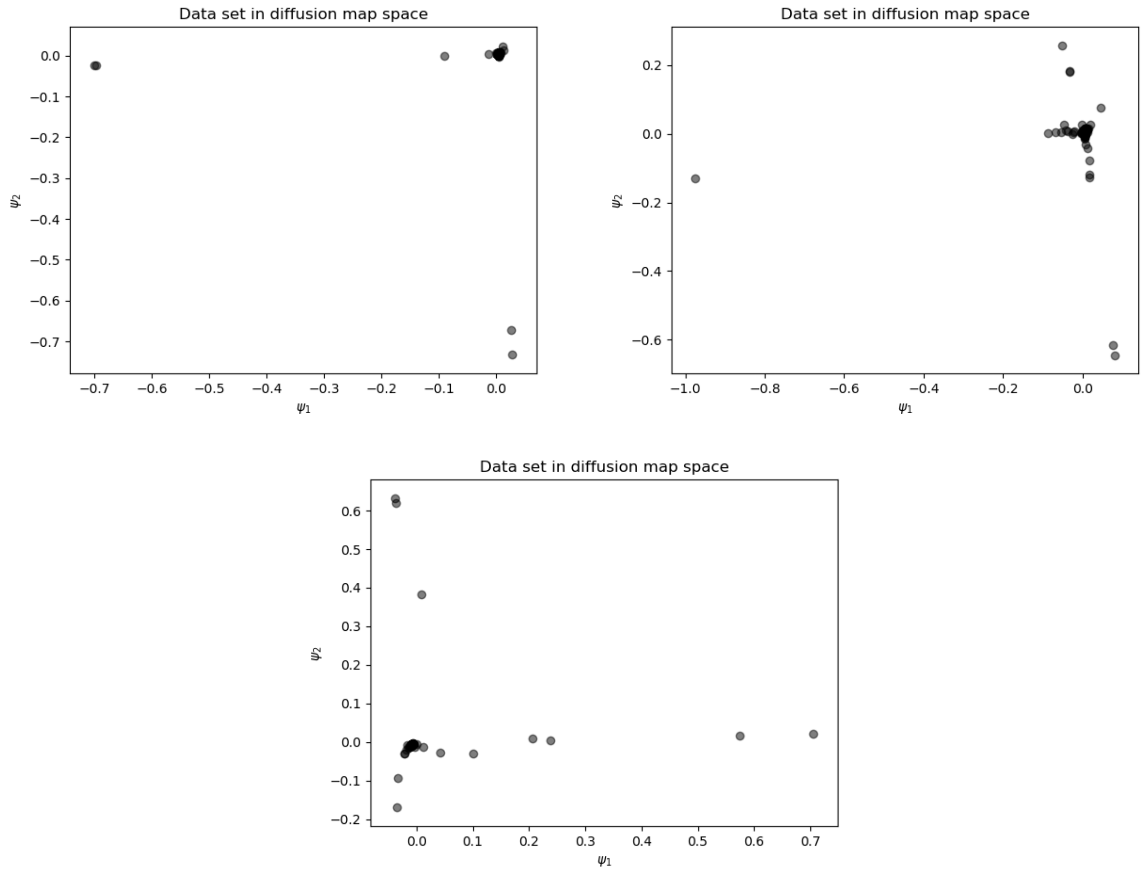


Figure 8: Projections of 2D XY data taken at $T/J = 0.87$ (top left), $T/J = .89$ (top right), and $T/J = 0.91$ (bottom) based on the Diffusion map algorithm.

5 Conclusion

The work presented above showcases the power of two different unsupervised learning algorithms to describe symmetry-breaking and topological phase transitions, respectively. While variational methods like the neural network ansatz perform well at ground state optimization and are readily generalizable to different Hamiltonian models, in my experience for this project the need to tune hyperparameters such as the sample size and the learning rate makes the method less robust than alternatives that only rely on raw data.

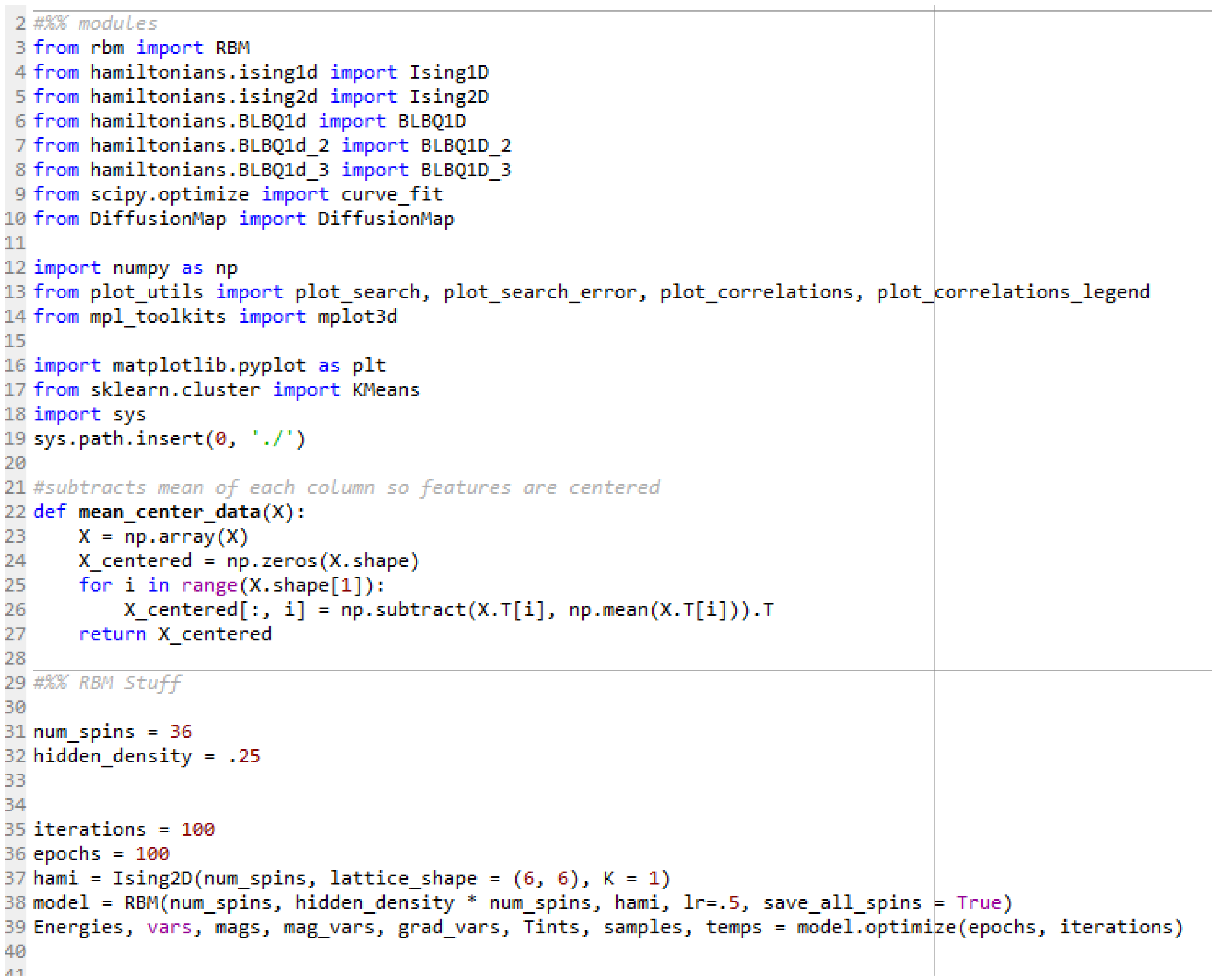
Unsupervised learning also lends itself more easily to the analysis of experimental data, due to the fact that the techniques are general to any data set on the manifold that it lives in. Knowledge of different algorithms' performances for different models acts as a guideline for which techniques to use on unknown data. As shown by the use of NQS data with PCA projection, unsupervised techniques can also act to benchmark the optimization of variational methods and act as preprocessing for training data with minimal loss of information. Understanding the success of diffusion maps for capturing global structure in the XY model and other theories with topological order [19] is crucial for interpreting the unique information geometries that form in such systems and eventually paving the way to designing exotic topological materials.

6 Code Availability

The code screenshotted below was self written in Python and includes all work for NQS simulation of the 2D Ising model, PCA analysis, and clustering of the projected data. The actual code is available upon request. The code used to generate samples from the XY model and the Diffusion map implementation were taken from the following github links:

https://github.com/zhevnerchuk/XY-model-Metropolis-Simulation

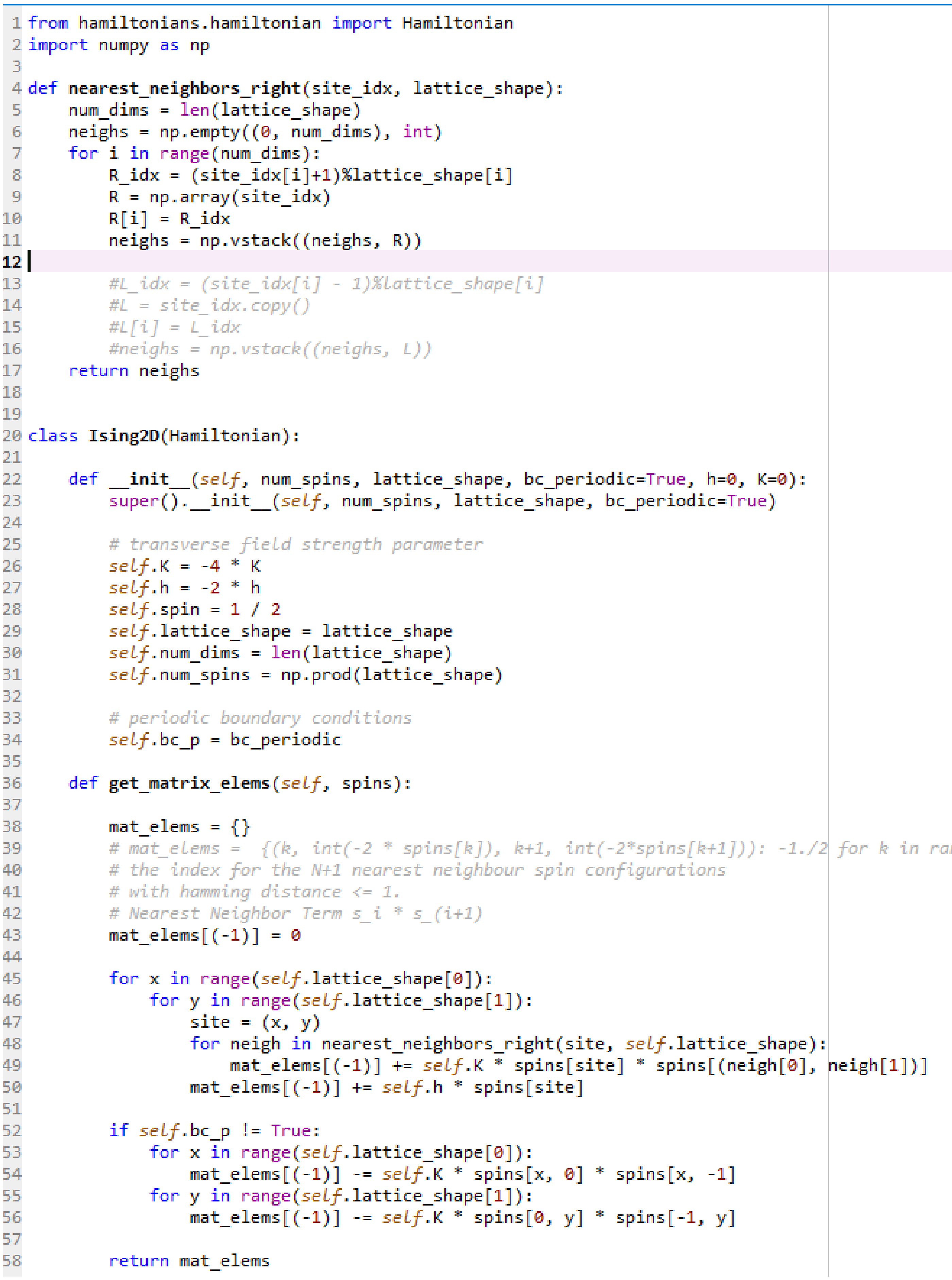
https://github.com/jmbr/diffusion-maps



```python
#%% modules
from rbm import RBM
from hamiltonians.ising1d import Ising1D
from hamiltonians.ising2d import Ising2D
from hamiltonians.BLBQ1d import BLBQ1D
from hamiltonians.BLBQ1d_2 import BLBQ1D_2
from hamiltonians.BLBQ1d_3 import BLBQ1D_3
from scipy.optimize import curve_fit
from DiffusionMap import DiffusionMap

import numpy as np
from plot_utils import plot_search, plot_search_error, plot_correlations, plot_correlations_legend
from mpl_toolkits import mplot3d

import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import sys
sys.path.insert(0, './')

#subtracts mean of each column so features are centered
def mean_center_data(X):
    X = np.array(X)
    X_centered = np.zeros(X.shape)
    for i in range(X.shape[1]):
        X_centered[:, i] = np.subtract(X.T[i], np.mean(X.T[i])).T
    return X_centered

#%% RBM Stuff

num_spins = 36
hidden_density = .25


iterations = 100
epochs = 100
hami = Ising2D(num_spins, lattice_shape = (6, 6), K = 1)
model = RBM(num_spins, hidden_density * num_spins, hami, lr=.5, save_all_spins = True)
Energies, vars, mags, mag_vars, grad_vars, Tints, samples, temps = model.optimize(epochs, iterations)

```

```python
plt.figure()
plt.plot(range(epochs), Energies)
plt.title('Energy vs. Iteration')
plt.xlabel('Iteration')
plt.ylabel('Energy')


plt.figure()
plt.plot(range(epochs), vars)
plt.xlabel('Iteration')
plt.ylabel('Variance in energy')
plt.title('Variance vs. Iteration')

plt.figure()
plt.plot(range(epochs), mags/max(mags))
plt.xlabel('Iteration')
plt.ylabel('magnetization')
plt.title('Magnetization vs. Iteration')


plt.figure()
plt.plot(range(epochs), mag_vars)
plt.xlabel('Iteration')
plt.ylabel('Variance of magnetization')
plt.title('Magnetization Variance vs. Iteration')


plt.figure()
plt.plot(range(epochs), grad_vars)
plt.xlabel('Iteration')
plt.ylabel('Variance of average gradient')
plt.title('gradient Variance vs. Iteration anti ')


plt.figure()
plt.plot(range(epochs)[:15], Tints[:15])
plt.xlabel('Iteration')
plt.ylabel('correlation time')
plt.title('correlation time vs. Iteration')



correlations = []

correlations = np.zeros(hami.lattice_shape)
for x in range(hami.lattice_shape[0]):
    for y in range(hami.lattice_shape[1]):
        corr = model.sampler.compute_correlations((0, 0), (x, y))
        correlations[(x, y)] = corr

plot_correlations(correlations)
```

```python
#%% PCA

X = mean_center_data(samples)
gram = np.matmul(X.T, X)
eigs, vecs = np.linalg.eig(gram)

eigpairs = [(np.abs(eigs[i]), vecs[:, i]) for i in range(len(eigs))]
eigpairs.sort(key=lambda x: x[0], reverse=True)

first = X @ eigpairs[0][1]
second = X @ eigpairs[1][1]

#%% PCA plotting

plt.figure()
plt.scatter(first, second, c=temps, cmap='YlOrRd')
plt.colorbar(label='iteration')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()


#3D
first = X @ eigpairs[0][1]
second = X @ eigpairs[1][1]
third = X @ eigpairs[2][1]

plt.figure()
ax = plt.axes(projection='3d')
ax.set_xlabel('P1')
ax.set_ylabel('P2')
ax.set_zlabel('P3')
p = ax.scatter3D(first, second, third, c=temps, cmap='YlOrRd')
plt.colorbar(p)
plt.show()


plt.figure()
plt.step(range(len(eigs)), [i[0] for i in eigpairs])
plt.xlabel('eigenvalue index')
plt.show()

#%% Clustering Analysis


clust = KMeans(n_clusters = 3)
clust.fit(np.column_stack((first, second)))

plt.figure()
plt.scatter(first, second, c=clust.labels_, cmap='YlOrRd')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()


```



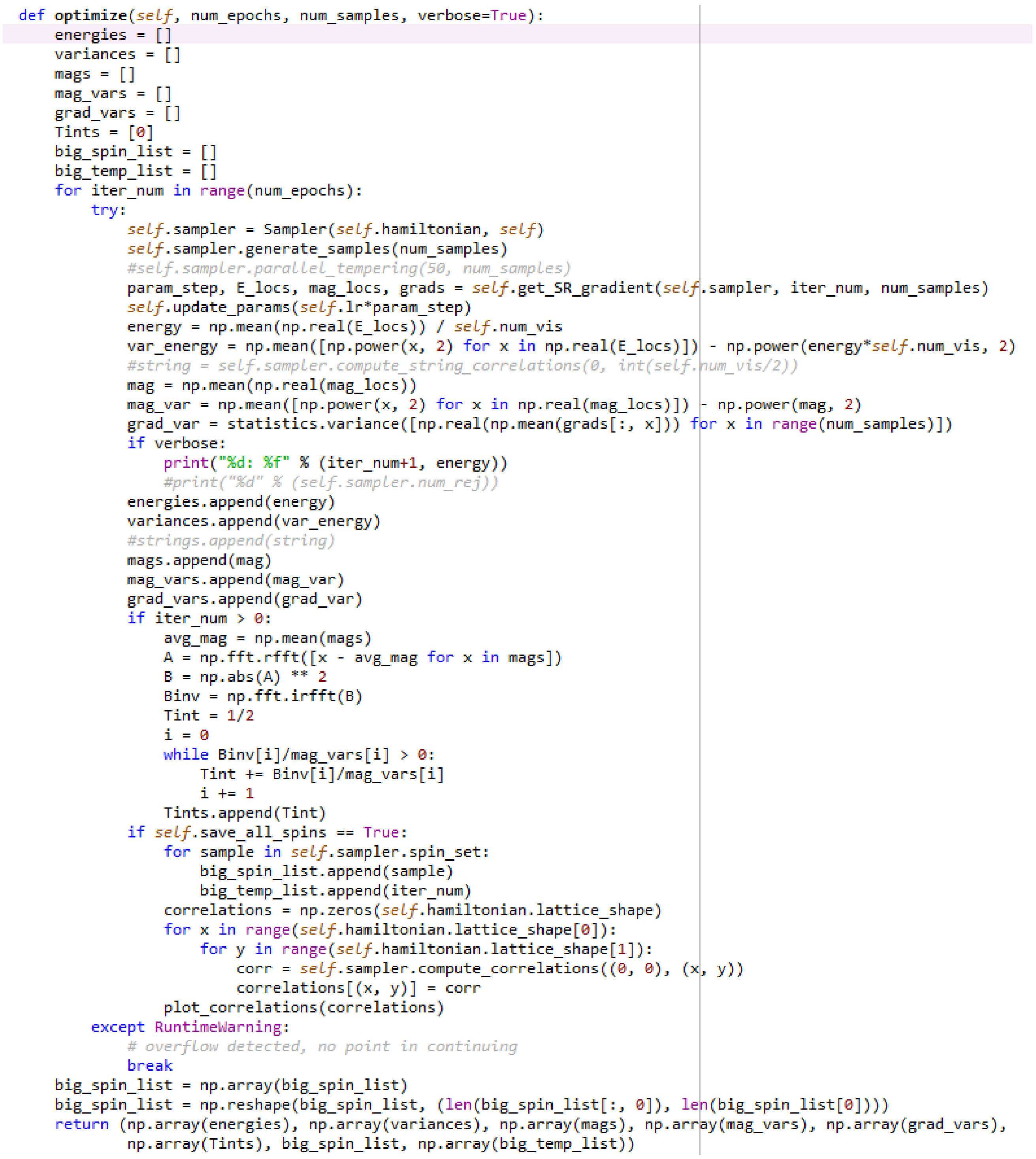
```python
from hamiltonians.hamiltonian import Hamiltonian
import numpy as np

def nearest_neighbors_right(site_idx, lattice_shape):
    num_dims = len(lattice_shape)
    neighs = np.empty((0, num_dims), int)
    for i in range(num_dims):
        R_idx = (site_idx[i]+1)%lattice_shape[i]
        R = np.array(site_idx)
        R[i] = R_idx
        neighs = np.vstack((neighs, R))

        #L_idx = (site_idx[i] - 1)%lattice_shape[i]
        #L = site_idx.copy()
        #L[i] = L_idx
        #neighs = np.vstack((neighs, L))
    return neighs


class Ising2D(Hamiltonian):

    def __init__(self, num_spins, lattice_shape, bc_periodic=True, h=0, K=0):
        super().__init__(self, num_spins, lattice_shape, bc_periodic=True)

        # transverse field strength parameter
        self.K = -4 * K
        self.h = -2 * h
        self.spin = 1 / 2
        self.lattice_shape = lattice_shape
        self.num_dims = len(lattice_shape)
        self.num_spins = np.prod(lattice_shape)

        # periodic boundary conditions
        self.bc_p = bc_periodic

    def get_matrix_elems(self, spins):

        mat_elems = {}
        # mat_elems =  {(k, int(-2 * spins[k]), k+1, int(-2*spins[k+1])): -1./2 for k in ra
        # the index for the N+1 nearest neighbour spin configurations
        # with hamming distance <= 1.
        # Nearest Neighbor Term s_i * s_(i+1)
        mat_elems[(-1)] = 0

        for x in range(self.lattice_shape[0]):
            for y in range(self.lattice_shape[1]):
                site = (x, y)
                for neigh in nearest_neighbors_right(site, self.lattice_shape):
                    mat_elems[(-1)] += self.K * spins[site] * spins[(neigh[0], neigh[1])]
                mat_elems[(-1)] += self.h * spins[site]

        if self.bc_p != True:
            for x in range(self.lattice_shape[0]):
                mat_elems[(-1)] -= self.K * spins[x, 0] * spins[x, -1]
            for y in range(self.lattice_shape[1]):
                mat_elems[(-1)] -= self.K * spins[0, y] * spins[-1, y]

        return mat_elems
```

```python
import numpy as np

def nearest_neighbors(site_idx, lattice_shape):
    site_idx = np.array(site_idx)
    num_dims = len(lattice_shape)
    neighs = np.empty((0, num_dims), int)
    for i in range(num_dims):
        R_idx = (site_idx[i]+1)%lattice_shape[i]
        R = site_idx.copy()
        R[i] = R_idx
        neighs = np.vstack((neighs, R))

        L_idx = (site_idx[i] - 1)%lattice_shape[i]
        L = site_idx.copy()
        L[i] = L_idx
        neighs = np.vstack((neighs, L))
    return neighs



class Sampler(object):
    """
    Performs Markov Chain Monte Carlo Sampling to generate a set of
    spin configurations sampled from the wave-function we are trying
    to model.
    """

    def __init__(self, hamiltonian, rbm, init_state=None, wolff = False):

        # hamiltonian of system to compute local energies
        self.hami = hamiltonian

        self.spin = self.hami.get_spin()

        # neural network spin model
        self.model = rbm

        self.wolff = wolff

        # number of spins
        self.num_spins = self.model.num_vis

        # list of spin configurations for computing energy gradients
        self.spin_set = []

        # list of magnetizations
        self.mag_set = []

        # list of local energies for each spin configuration in spin_set
        self.local_energies = []

        # number of rejected moves for each Metropolis sampling step
        self.num_rej = 0

        # current state during sampling process
        self.curr_state = init_state
        if self.curr_state is None:
            self.curr_state = self.init_random_state()
```

```python
def init_random_state(self):
    spins = np.random.choice(np.linspace(-1*self.spin, self.spin, 2*self.spin+1), self.hami.lattice_shape)
    return spins

def get_local_energy(self):
    mat_elems = self.hami.get_matrix_elems(self.curr_state)
    energy = 0
    for state_id, H_val in mat_elems.items():
        spins_prime = self.flip_spins(self.curr_state, state_id)
        term = self.model.amp_ratio(self.curr_state, spins_prime) * H_val
        energy += term
    return energy

def step(self, temp, num_flips=1):
    flip_idx = ()
    for i in range(num_flips):
            spins = self.curr_state
            flip_x = np.random.randint(0, self.hami.lattice_shape[0])
            flip_y = np.random.randint(0, self.hami.lattice_shape[1])
            if self.hami.get_spin() == .5:
                step = -2 * spins[flip_x, flip_y]
            else:
                step = 2*np.random.binomial(1, .5) - 1
            flip_idx += ((flip_x, flip_y), step)
            [accepted, spins_prime] = self.flip_accepted(spins, flip_idx, temp)

            if accepted:
                self.model.update_eff_angles(spins, spins_prime)
                self.curr_state = spins_prime
            else:
                self.num_rej += 1
# returns whether the flipped state is accepted and the flipped state
def flip_accepted(self, spins, flip_idx, temp):
    spins_prime = spins.copy()
    for i in range(0, len(flip_idx), 2):
        spins_prime[flip_idx[i]] += flip_idx[i + 1]
        if abs(spins_prime[flip_idx[i]]) > self.spin:
            return [False, 0]
    transition_prob = np.power(np.absolute(self.model.amp_ratio(spins, spins_prime)), 2./temp)
    return [transition_prob >= np.random.random(), spins_prime]

def flip_spins(self, spins, flip_idx):
    spins_prime = spins.copy()
    if flip_idx == -1:
        return spins_prime
    else:
        L = len(flip_idx)
        for i in range(0, L, 2):
            spins_prime[0, flip_idx[i]] = np.add(spins_prime[0, flip_idx[i]], flip_idx[i + 1], casting="unsafe")
        return spins_prime
def generate_samples(self, num_samples, temp = 1):
    self.model.init_effective_angles(self.curr_state)
    # Monte Carlo Sampling
    for i in range(int(num_samples)):
        for k in range(int(1*self.model.num_var)):
            self.step(temp)
        self.spin_set.append(self.model.mat2vec(self.curr_state))
        self.local_energies.append(self.get_local_energy())
        self.mag_set.append(2*np.mean(self.curr_state))
```

```python
import numpy as np
from sampler import Sampler
import statistics

from plot_utils import plot_search, plot_search_error, plot_correlations, plot_correlations_legend


from MinresQLP import Minresqlp

def sigmoid(x):
    return np.power((1 + np.exp(-x)), -1)

class RBM(object):
    def __init__(self, num_vis, num_hid, hami, lr = 1, save_all_spins = False):

        self.save_all_spins = save_all_spins
        # number of visible units (spins in the system)
        self.num_vis = int(num_vis)

        # number of hidden units (hidden correlation parameters)
        self.num_hid = int(num_hid)

        # number of model parameters
        self.num_var = self.num_vis + self.num_hid + self.num_vis*self.num_hid

        # bias on visible units
        self.vis_bias = self.init_weights((self.num_vis, 1))

        # bias on hidden units
        self.hid_bias = self.init_weights((self.num_hid, 1))

        # weight matrix
        self.W = self.init_weights((self.num_vis, self.num_hid))

        # look-up tables for efficient computation of wave function amplitudes
        self.effective_angles = np.complex128(np.zeros((self.num_hid, 1)))

        # hamiltonian governing evolution of system
        self.hamiltonian = hami

        # learning rate
        self.lr = lr

        self.sampler = None

    # Initialize matrix of complex numbers within a range set below.
    def init_weights(self, size):
        if type(size) == tuple:
            size = list(map(int, size))
        else:
            size = int(size)
        a = (np.random.normal(0, .01, size) - .5) * 10e-2
        b = (np.random.normal(0, .01, size) - .5) * 10e-2
        return np.complex128(a + b * 1j)

    def mat2vec(self, spins):
        return spins.reshape(self.num_vis, 1)

```

```python
def vec2mat(self, vec):
    return np.reshape(vec, self.hamiltonian.lattice_shape)

# Initialize effective angles with a given spin.
def init_effective_angles(self, spins):
    spin_list = self.mat2vec(spins)
    self.effective_angles = self.hid_bias + self.W.T @ spin_list

# update look-up tables when new spin configuration accepted by Metropolis Algorithm
def update_eff_angles(self, spins, spins_prime):
    spins_diff = spins - spins_prime
    self.effective_angles -= self.W.T @ (spins_diff).reshape((self.num_vis, 1))

# returns wave function amplitude of SPINS configuration
def Psi_M(self, spins):
    return np.complex128(np.exp(np.dot(self.vis_bias.T, spins)) * np.prod(2 * np.cosh(self.effective_angles)))

# ratio of wave function amplitudes for spin configurations spins_prime/spins
def amp_ratio(self, spins, spins_prime):
    return np.exp(self.log_amp_ratio(spins, spins_prime))

# logarithm of amplitude ratio (easier for numerical stability reasons) log(psi(spins_prime)/psi(spins))
def log_amp_ratio(self, spins, spins_prime):
    spins_diff = self.mat2vec(spins - spins_prime)
    piece1 = -1*np.dot(self.vis_bias.T, spins_diff)
    piece2 = np.sum(np.log(np.cosh(self.effective_angles - self.W.T @ spins_diff))
                    - np.log(np.cosh(self.effective_angles)))
    log = piece1 + piece2
    return log
```



```python
def optimize(self, num_epochs, num_samples, verbose=True):
    energies = []
    variances = []
    mags = []
    mag_vars = []
    grad_vars = []
    Tints = [0]
    big_spin_list = []
    big_temp_list = []
    for iter_num in range(num_epochs):
        try:
            self.sampler = Sampler(self.hamiltonian, self)
            self.sampler.generate_samples(num_samples)
            #self.sampler.parallel_tempering(50, num_samples)
            param_step, E_locs, mag_locs, grads = self.get_SR_gradient(self.sampler, iter_num, num_samples)
            self.update_params(self.lr*param_step)
            energy = np.mean(np.real(E_locs)) / self.num_vis
            var_energy = np.mean([np.power(x, 2) for x in np.real(E_locs)]) - np.power(energy*self.num_vis, 2)
            #string = self.sampler.compute_string_correlations(0, int(self.num_vis/2))
            mag = np.mean(np.real(mag_locs))
            mag_var = np.mean([np.power(x, 2) for x in np.real(mag_locs)]) - np.power(mag, 2)
            grad_var = statistics.variance([np.real(np.mean(grads[:, x])) for x in range(num_samples)])
            if verbose:
                print("%d: %f" % (iter_num+1, energy))
                #print("%d" % (self.sampler.num_rej))
            energies.append(energy)
            variances.append(var_energy)
            #strings.append(string)
            mags.append(mag)
            mag_vars.append(mag_var)
            grad_vars.append(grad_var)
            if iter_num > 0:
                avg_mag = np.mean(mags)
                A = np.fft.rfft([x - avg_mag for x in mags])
                B = np.abs(A) ** 2
                Binv = np.fft.irfft(B)
                Tint = 1/2
                i = 0
                while Binv[i]/mag_vars[i] > 0:
                    Tint += Binv[i]/mag_vars[i]
                    i += 1
                Tints.append(Tint)
            if self.save_all_spins == True:
                for sample in self.sampler.spin_set:
                    big_spin_list.append(sample)
                    big_temp_list.append(iter_num)
                correlations = np.zeros(self.hamiltonian.lattice_shape)
                for x in range(self.hamiltonian.lattice_shape[0]):
                    for y in range(self.hamiltonian.lattice_shape[1]):
                        corr = self.sampler.compute_correlations((0, 0), (x, y))
                        correlations[(x, y)] = corr
                plot_correlations(correlations)
        except RuntimeWarning:
            # overflow detected, no point in continuing
            break
    big_spin_list = np.array(big_spin_list)
    big_spin_list = np.reshape(big_spin_list, (len(big_spin_list[:, 0]), len(big_spin_list[0])))
    return (np.array(energies), np.array(variances), np.array(mags), np.array(mag_vars), np.array(grad_vars),
            np.array(Tints), big_spin_list, np.array(big_temp_list))
```

```
# returns the SR parameter update using the samples generated by SAMPLER
#  as well as local energies for each spin configuration created
def get_SR_gradient(self, sampler, iter_num, num_samples):
    spin_set = np.array(sampler.spin_set).reshape((num_samples, self.num_vis))
    E_locs = np.array(sampler.local_energies).reshape((spin_set.shape[0], 1))
    mag_locs = np.array(sampler.mag_set)
    param_step, grads = self.compute_grads(iter_num, spin_set, E_locs, num_samples)
    return param_step, E_locs, mag_locs, grads

# Computes variational parameters and returns the parameter update for each weight in
# the RBM as well as the gradients calculated
def compute_grads(self, iter_num, spin_set, E_locs, num_samples):
    eff_angles = self.W.T @ spin_set.T + self.hid_bias
    var_a = np.reshape(spin_set.T, (self.num_vis, num_samples))
    var_b = np.tanh(eff_angles)
    var_b = np.reshape(var_b, (self.num_hid, num_samples))
    var_W = (spin_set.T).reshape((self.num_vis, 1, num_samples)) * np.tanh(eff_angles.reshape((1, self.num_hid, num_samples)))

    grads = np.concatenate([var_a, var_b, var_W.reshape(self.num_vis*self.num_hid, num_samples)])

    exp_grads = np.real(np.sum(grads, axis=1, keepdims=True)) / num_samples

    # Cross-correlation matrix S
    exp_grads_matrix = np.conj(exp_grads.reshape((grads.shape[0], 1))) * exp_grads.reshape((1, grads.shape[0]))
    cross_term = (np.conjugate(grads) @ grads.T)/num_samples
    S = cross_term - exp_grads_matrix

    F = np.sum(E_locs.T * grads.conj(), axis=1)/num_samples
    F -= np.sum(E_locs.T, axis=1) * np.sum(grads.conj(), axis=1) /np.square(num_samples)
    lamb = 100*(.9**iter_num)
    if lamb < 10e-4:
        lamb = 10e-4
    S_ridge = S + lamb*np.eye(S.shape[0])
    lr = self.lr/np.sqrt(iter_num+1)
    param_step = Minresqlp(S_ridge, -1*lr*F, rtol = 1e-8, maxit = self.num_var)[0]

    return param_step, grads

# updates each weight parameter based on the output of COMPUTE_GRADS
def update_params(self, steps):
    self.vis_bias += steps[:self.num_vis].copy()
    self.hid_bias += steps[self.num_vis:(self.num_vis+self.num_hid)].copy()
    self.W += steps[(self.num_vis+self.num_hid):].reshape((self.num_vis, self.num_hid)).copy()
```

References

[1] Matthew J. S. Beach, Anna Golubeva, and Roger G. Melko. "Machine learning vortices at the Kosterlitz-Thouless transition". In: *Physical Review B* 97.4 (2018). DOI: 10.1103/physrevb.97.045207.

[2] Carleo et al. *Solving the Quantum Many-Body Problem with Artificial Neural Networks*. June 2016. URL: https://arxiv.org/abs/1606.02318.

[3] Jing Chen et al. "Equivalence of restricted Boltzmann machines and tensor network states". In: *Physical Review B* 97.8 (2018). DOI: 10.1103/physrevb.97.085104.

[4] Ronald R. Coifman and Stéphane Lafon. "Diffusion maps". In: *Applied and Computational Harmonic Analysis* 21.1 (2006), pp. 5–30. DOI: 10.1016/j.acha.2006.04.006.

[5] Elbio Dagotto. "Correlated electrons in high-temperature superconductors". In: *Reviews of Modern Physics* 66.3 (1994), pp. 763–840. DOI: 10.1103/revmodphys.66.763.

[6] Deng et al. *Quantum Entanglement in Neural Network States*. May 2017. URL: https://arxiv.org/abs/1701.04844.

[7] Ivan Glasser et al. "Neural-Network Quantum States, String-Bond States, and Chiral Topological States". In: *Physical Review X* 8.1 (2018). DOI: 10.1103/physrevx.8.011006.

[8] M B Hastings. "An area law for one-dimensional quantum systems". In: *Journal of Statistical Mechanics: Theory and Experiment* 2007.08 (2007). DOI: 10.1088/1742-5468/2007/08/p08024.

[9] Geoffrey E. Hinton. "A Practical Guide to Training Restricted Boltzmann Machines". In: *Lecture Notes in Computer Science Neural Networks: Tricks of the Trade* (2012), pp. 599–619. DOI: 10.1007/978-3-642-35289-8_32.

[10] Wenjian Hu, Rajiv R. P. Singh, and Richard T. Scalettar. "Discovering phases, phase transitions, and crossovers through unsupervised machine learning: A critical examination". In: *Physical Review E* 95.6 (2017). DOI: 10.1103/physreve.95.062122.

[11] Huang et al. *Neural network representation of tensor network and chiral states*. Jan. 2017. URL: https://arxiv.org/abs/1701.06246.

[12] B. Keimer and J. E. Moore. “The physics of quantum materials”. In: *Nature Physics* 13.11 (2017), pp. 1045–1055. DOI: 10.1038/nphys4302.

[13] J M Kosterlitz and D J Thouless. “Ordering, metastability and phase transitions in two-dimensional systems”. In: *Journal of Physics C: Solid State Physics* 6.7 (Dec. 1973), pp. 1181–1203. DOI: 10.1088/0022-3719/6/7/010.

[14] R. B. Laughlin. “Anomalous Quantum Hall Effect: An Incompressible Quantum Fluid with Fractionally Charged Excitations”. In: *Physical Review Letters* 50.18 (1983), pp. 1395–1398. DOI: 10.1103/physrevlett.50.1395.

[15] Sirui Lu, Xun Gao, and L.-M. Duan. “Efficient representation of topologically ordered states with restricted Boltzmann machines”. In: *Physical Review B* 99.15 (2019). DOI: 10.1103/physrevb.99.155136.

[16] W. L. Mcmillan. “Ground State of LiquidHe4”. In: *Physical Review* 138.2A (1965). DOI: 10.1103/physrev.138.a442.

[17] Román Orús. “A practical introduction to tensor networks: Matrix product states and projected entangled pair states”. In: *Annals of Physics* 349 (2014), pp. 117–158. DOI: 10.1016/j.aop.2014.06.013.

[18] Karl Pearson. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[19] Joaquin F. Rodriguez-Nieva and Mathias S. Scheurer. “Identifying topological order through unsupervised machine learning”. In: *Nature Physics* 15.8 (June 2019), pp. 790–795. DOI: 10.1038/s41567-019-0512-x.

[20] Hiroki Saito and Masaya Kato. “Machine Learning Technique to Find Quantum Many-Body Ground States of Bosons on a Lattice”. In: *Journal of the Physical Society of Japan* 87.1 (2018), p. 014001. DOI: 10.7566/jpsj.87.014001.

[21] K. Sohara and M. Kotani. “Application of Kernel Principal Components Analysis to pattern recognitions”. In: *Proceedings of the 41st SICE Annual Conference. SICE 2002.* (). DOI: 10.1109/sice.2002.1195250.

[22] Lei Wang. “Discovering phase transitions with unsupervised learning”. In: *Physical Review B* 94.19 (Feb. 2016). DOI: 10.1103/physrevb.94.195105.

[23] Pengfei Zhang, Huitao Shen, and Hui Zhai. “Machine Learning Topological Invariants with Neural Networks”. In: *Physical Review Letters* 120.6 (June 2018). DOI: 10.1103/physrevlett.120.066401.