February 2022

Ph.D. Dissertation

Accelerated Dynamics Simulation using Deep Learning Corrections

Graduate School of Chosun University

Department of Physics

Park Sojeong

Accelerated Dynamics Simulation using Deep Learning Corrections

딥러닝 보정을 이용한 동력학 시뮬레이션의 가속화 연구

February 25, 2022

Graduate School of Chosun University

Department of Physics

Park Sojeong

Accelerated Dynamics Simulation using Deep Learning Corrections

Advisor Kwak Wooseop

Co-advisor Lee Hwee Kuan

This thesis is submitted to the Graduate School of Chosun University in partial fulfilment of the requirements for the Doctor's degree in physics.

October 2021

Graduate School of Chosun University

Department of Physics

Park Sojeong

This certifies that Ph.D. Dissertation

of Park Sojeong is approved.

최 은 서 (sign)

Thesis Committee Chair: Prof. Choi Eun Seo

곽 우 섭 (sign)

Thesis Committee Member: Prof. Kwak Wooseop

Lee Hwee Kuan (sign)

Thesis Committee Member: Prof. Lee Hwee Kuan

류 설 (sign)

Thesis Committee Member: Prof. Ryu Seol

김 승 연 (sign)

Thesis Committee Member: Prof. Kim Seung Yeon

January 2022

Graduate School of Chosun University

Contents

List of Figures

Abstract

Accelerated Dynamics Simulation using Deep Learning Corrections

Park Sojeong

Advisor Kwak Wooseop

Co-advisor Lee Hwee Kuan

Department of Physics

Graduate School of the Chosun University

Theoretical models capture very precisely the behaviour of materials at the molecular and atomic level. This makes computer simulations, such as spin dynamics simulations of magnetic materials and molecular dynamic simulation for analyzing the physical movements of atoms and molecules, accurately mimic experimental results. New approaches to efficient computer simulations are limited by integration time step barrier to solving the equations of motions of many-body problems. Using a short time step leads to an accurate but inefficient simulation regime whereas using a large time step leads to accumulation of numerical errors that render the whole simulation useless.

In the study of spin dynamic simulations, we use a Deep Learning method to compute the numerical errors of each large time step and use these computed errors to make corrections to achieve higher accuracy in our spin dynamics. We validate our method on the 3D Ferromagnetic Heisenberg cubic lattice over a range of temperatures. Here we show that the Deep Learning method can accelerate the simulation speed by 10 times while maintaining simulation accuracy and overcome the limitations of requiring small time steps in spin dynamic simulations.

In the study of molecular dynamics, Deep Learning is used to learn the effective forces governed by pair-wise interactions on the Lennard-Jones system. Once many body or two body interactions have been learned, a single neural network model is used to perform accelerated simulations for the Hamiltonian system of interest for arbitrary thermodynamics states. We show that by using large time step, our Deep Learning method recovers the stability of simulation by preventing overflow due to artificially large force and shows 10 times speed up of simulation by producing similar simulation accuracy.

Keyword: Spin dynamics simulations, Molecular dynamics simulations, Deep learning, Heisenberg models, Lennard-Jones system

국문초록

딥러닝 보정을 이용한 동력학 시뮬레이션의 가속화 연구

박 소 정

지도교수 곽 우 섭

공동지도교수 Lee Hwee Kuan

조선대학교 대학원 물리학과

이론적인 모델은 분자 및 원자 수준에서 물질의 움직임을 매우 정확하게 연구되어진다. 따라서 자성 물질의 스핀 동역학 시뮬레이션 및 원자 및 분자의 물리적 움직임을 분석하기 위한 분자 동력학 시뮬레이션과 같은 컴퓨터 시뮬레이션이 실험 결과를 정확하게 모방한다. 효율적인 컴퓨터 시뮬레이션에 대한 새로운 접근 방식은 다체 문제의 운동 방정식을 풀기 위한 시간 적분 간격의 장벽으로 인해 적분 간격이 제한된다. 짧은 시간 간격을 사용하면 정확하지만 비효율적인 시뮬레이션 체제가 생성되는 반면, 큰 시간 간격을 사용하면 전체 시뮬레이션의 수치적 오차가 누적된다.

스핀 동역학 시뮬레이션 연구에서 우리는 다양한 온도 범위에서 3D 입방 격자 안에 있는 강자성 Heisenberg 스핀 모형으로 딥 러닝 방법을 사용하여 각 큰 시간 간격의 수치적 오차를 계산하고 이러한 계산된 오차를 사용하여 스핀 동역학 시뮬레이션에서 더 높은 정확도를 달성한다. 여기에서 딥 러닝 방법이 시

뮬레이션 정확도를 유지하면서 시뮬레이션 속도를 10배 가속화할 수 있고 스핀 동역학 시뮬레이션에서 작은 시간 간격의 한계를 극복할 수 있음을 보여준다. 분자 동역학 연구에서 딥 러닝 방법은 레너드 존스 상호 작용에 의한 시스템에서 근본적인 힘을 학습하여 딥러닝 보정을 통해 분자 동역학 결과를 향상시키는데 사용한다. 두 물체 또는 다체 간의 상호작용이 학습되어지면, 뉴럴 네트워크 모델을 사용하여 임의의 열역학 상태에서 해밀턴 시스템에 대한 가속 시뮬레이션을 수행한다. 딥 러닝 방법을 이용하여 두 원자 간의 간격이 가까워지면 생기는 컴퓨터 시뮬레이션의 오퍼플로우 문제를 해결함으로써 컴퓨터 시뮬레이션의 안정성을 개선하고 분자 동력학 시뮬레이션의 정확도를 유지함과 동시에 속도를 약 10배 향상시킴을 보여준다.

주요어휘: 스핀 동력학 시뮬레이션, 분자 동력학 시뮬레이션, 딥 러닝, 하이젠베르크 모형, 레너드 존스 시스템

Chapter 1

Introduction

The computer simulation methods are powerful tools for understanding fundamental properties of materials that can be verified by experimental methods. There are two methods in computer simulations, spin dynamics simulation and molecular dynamics simulations. These methods have been used to investigate the time evolution of physical quantities of physical systems under different conditions. Spin dynamics simulations [1] have been widely used to study the underlying physics of magnetic material performed experimentally by using neutron scattering [2]. The study of the properties of these magnetic materials enables us to develop much better applications such as in Nd-Fe-B-type permanent magnets used for motors in hybrid cars [3, 4], magnetoresistive random access memory (MRAM) based on the storage of data in stable magnetic states [5], ultrafast spins dynamics in magnetic nanostructures [6, 7], heat assisted magnetic recording and ferromagnetic reso-

nance methods for increasing the storage density of hard disk drives [8, 9], exchange bias related to magnetic recording [10], and magnetocaloric materials for refrigeration technologies [3]. The molecular dynamics simulations [11, 12] have been used to the study for investigating the time evolution of atomic and molecular systems such as Lennard-Jones fluid [13, 14] and have a wide range of applications to the design of new materials, drug design [15], and protein folding [16].

In computer simulations, classical equations of motion of dynamic systems are solved numerically using well known integrators such as leapfrog, Verlet, predictor-corrector, and Runge-Kutta methods [17, 18, 19]. The accuracy of these simulations depends on a time integration step size. If a large time step is used, the accumulated truncation error becomes larger. Conversely, using a short time step is very computationally demanding. So, it is important to find a trade off between speed and accuracy.

Symplectic methods [20, 21] are among the most useful time integrators for dynamics simulations. The numerical solutions of symplectic methods have properties of the time reversibility, conservation of the phase-space volume exactly, and the error in the total energy of the system bounded. For example, high order Suzuki-Trotter decomposition method, one of the symplectic methods, allows for larger time step with limited error in its computation. We seek to enhance the time integration step of symplectic methods further using Deep Learning techniques.

CHAPTER 1. INTRODUCTION

Machine learning techniques including Deep learning are used to reduce the computational cost compared with the often high cost of numerical simulations. Recently, Machine Learning techniques are used to enhance simulation efficiencies in the condensed matter physics. Its applications include addressing difficulties of phase transition [22, 23, 24, 25, 26, 27] and accelerating the Monte-Carlo simulations [28]. A crucial issue in molecular dynamics simulations [29] is that generating samples from the equilibrium distributions is time consuming. Boltzmann generators machine [30] addresses the long-standing rare-event (e.g. transition) sampling problem. In addition, study of quantum many body systems using Machine Learning is applied to simulation of the quantum spin dynamics [31, 32], identifying phase transitions [33] and solving the exponential complexity of the many body problem in quantum systems [34].

We show that speed up is achieved if Deep Learning learns the corrections terms. The first condition for speed up is enough capacity of Deep Learning to learn the associations between spin (or molecule) configuration generated by large time steps and spin (or molecule) configuration generated by accurate short time steps. The second condition is enough training data for learning and show the Deep Learning enough pairs of patterns between spin (or molecule) configuration for large and short time steps.

In spin dynamics, we propose to use Deep Learning to estimate the correction terms of Suzuki-Trotter decomposition method, and then add the

correction terms back to spin dynamics simulations results, making them more accurate [35]. As a result of this correction, larger time step can be used for Suzuki-Trotter decomposition method, and corrections can be made for each time step. To evaluate our Deep Learning method, we analyze spin-spin correlation as a more stringent measure. We also use thermal averages to benchmark the performance of our method. We compare the Deep Learning results with those from spin dynamics simulations without Deep Learning for short time steps.

In molecular dynamics, the large time step causes the numerical integrator to become unstable and particles can overlap, causing artificially large force calculation. To recover the stability of simulation, our Deep Learning method is used to learn the effective forces acting on particles. We develop our neural networks to learn effective many-body interactions replacing the force calculations from the original physical model ($-\frac{\partial H}{\partial \mathbf{q}_\alpha}$). Once a new two body and many-body interactions have been learned, a single neural network model can be used to perform accelerated simulations for the Hamiltonian system of interest for arbitrary thermodynamics states (e.g. different temperatures, pressures, number of particles, and densities).

Chapter 2

Theoretical Background

Computer simulations have been used for a broad range of phenomena in statistical physics. Machine learning (ML) techniques including deep learning (DL) are applied in enhancing the performance of computer simulations of materials in physical dynamical systems. This chapter presents the theoretical framework for computer simulations and DL techniques.

2.1 Statistical Mechanics for Simulations

This section introduces Metropolis Monte Carlo method and understands the phase transitions in systems including magnets and liquids.

2.1.1 The Markov Chain and Detailed Balance

$$\frac{dP(x)}{dt} = \sum_{x'} T(x' \to x)P(x') - \sum_{x'} T(x \to x')P(x) \tag{2.1}$$

The master equation considers the change of the probability with time t, expressing the fact that $\sum_x P(x) \equiv 1$ at all times. All probability of state x' that is lost by a move away from the state x is gained in the probability of that state. The detailed balance with the equilibrium probability $P_{eq}(x)$ is

$$\begin{aligned} T(x' \to x)P_{eq}(x') &= T(x \to x')P_{eq}(x) \\ \frac{dP_{eq}(x)}{dt} &= 0, \text{ In equilibrium} \end{aligned} \tag{2.2}$$

2.1.2 Statistical Ensembles

In statistical mechanics, the behavior of a system based on the possible microstates are interested and known as the ensemble of states for a system.

Microcanonical Ensemble of Systems

The microcanonical ensemble is an isolated system with a number of particle N, volume V and energy U fixed. The probability of the system being in a certain microstate k is

$$P_k = \frac{1}{\Omega(U, V, N)} \tag{2.3}$$

where $\Omega = \sum_k 1$ is defined as the total number of microstates k for the system. This probability and thermodynamics are related through the Boltzmann relation and the thermodynamic potential to describe the isolated sys-

tem is entropy, S=S(U,V) which is maximum at equilibrium state when the energy states are discrete in Eq. (2.4).

$$S = k_B ln\Omega \approx k_B ln W_{max} \tag{2.4}$$

where $W = \frac{N!}{n_1!n_2!...}$, n_i is the number of particles in the energy E_i, and W_{max} is maximum number of microstates. The Boltzmann probability distribution is derived by using the principle of maximum entropy which requires N has to be large and maximize entropy by inserting Largrange multipliers. The probabilty of it being with energy E_k in the energy level k is

$$P_k = \frac{1}{Z} e^{-\beta E_k} \tag{2.5}$$

where $Z = \sum_k e^{-\beta E_k}$ and $\beta = \frac{1}{k_B T}$.

Canonical Ensemble of Systems

The exchange of heat between the system and its surroundings brings the constant temperature at the equilibrium state. In the canonical ensemble, a set of microstates with a number of particle N, volume V and temperature T is fixed but variable energy U. The partition function for a classical system is

$$Z(T,V,N) = \sum_{microstates\ k} e^{-\beta E_k} = \sum_{energies\ i} g_i e^{-\beta E_i} \tag{2.6}$$

where the first sum is over all possible states k with energy E_k, $\beta = 1/k_B T$ with k_B Boltzmann's constant, and the second sum is all possible states i with energy E_i, g_i is degeneracy which is the number of microstates with

E_i. The probability of any state of the system is determined by the partition function. The probability of finding the k^{th} state with energy E_k is given by

$$P_k = \frac{1}{Z} e^{-\beta E_k} \tag{2.7}$$

where the exponential factor is called the Boltzmann factor. All macroscopic thermodynamic properties are connected with the partition function. The ensemble average of a thermodynamic quantity A can be calculated using the canonical probability distribution.

$$< A >= \sum_{k}^{M} A_k P_k = \frac{\sum_k^M A_k e^{-\beta E_k}}{\sum_k^M e^{-\beta E_k}} \tag{2.8}$$

where M is the total number of accessible microstates.

2.1.3 Monte Carlo Move

The Metropolis Monte Carlo method is applied for evolution in the canonical ensemble. The evolution is driven by the energy change between the old and new configuration, $\Delta E = E_{new} - E_{old}$. From the detailed balance in Eq. (2.2), $T(x \to x')$ is the transition probability for trial move $x \to x'$. The relative probability is the ratio of the individual probability in Eq. (2.9).

$$\frac{T(x \to x')}{T(x' \to x)} = \frac{P(x')}{P(x)} = \frac{e^{-\beta E_{x'}}}{e^{-\beta E_x}} \tag{2.9}$$

The transition probability $T(x \to x')$ itself is used for acceptance probability calculation by Metropolies et al [36].

$$T(x \to x') = min(1, e^{-\beta \Delta E}) \tag{2.10}$$

where $\Delta E = E_{x'} - E_x$ is the energy change by the move from x to x'. The Metropolis Monte Carlo method is used for calculation of ensemble averages with importance sampling. Eq. (2.11) represents that only a finite number m of the total number of all microstates M in Eq. (2.8) are generated as follows.

$$< A > \approx A_m = \frac{\sum_k^m A_k e^{-\beta E_k}}{\sum_k^m e^{-\beta E_k}} = \sum_k^m A_k \pi_k \tag{2.11}$$

The m configurations are generated with random distribution π_k and then make a simple average of A as follows.

$$< A >= \frac{1}{m} \sum_k^m A_k \tag{2.12}$$

2.1.4 Phase Transitions

Phase transitions are boundaries between different phases of matter, eg. liquid-gas in fluids, order-disorder in magnet. The critical temperature T_c can be estimated by specific heat per spin or particle N that is computed from the fluctuations of the internal energy U in eq. (2.13)

$$c = \frac{\beta^2}{N}(< U^2 > - < U >^2) \tag{2.13}$$

where $< U >$ is expressed in terms of the thermal average of the energy. The fluctuations are intrinsic to the system evolution and large near the phase transition.

Tree-dimensional Classical Heisenberg Ferromagnet

The Hamiltonian for the ferromagnetic Heisenberg model on a cubic lattice is given by the hamiltonian as follow:

$$H = -J \sum_{<i,j>} \boldsymbol{S}^i \cdot \boldsymbol{S}^j \tag{2.14}$$

where a vector $\boldsymbol{S}^i$ has three components (S^i_x, S^i_y, S^i_z) and $|\boldsymbol{S}^i|$ is a unit vector. This model undergoes a phase transition at a temperature $k_B T_c/J = 1.442\ldots$ [37], where k_B is Boltzmann's constant.

Two-dimensional Lennard-Jones Systems

Given a vector of positions $\vec{q} = (q_1, q_2, ...)$ of n particles in a d dimension space, we have nd dimension, consider a potential ϕ that maps this nd dimension into $\mathbb{R}$. Consider the Lennard-Jones potential for two particles,

$$\phi(q_1, q_2) = 4\epsilon \left(\frac{\sigma^{12}}{|q_1 - q_2|^{12}} - \frac{\sigma^6}{|q_1 - q_2|^6} \right) \tag{2.15}$$

here ϵ is depth of $\phi(q_1, q_2)$ at the minimum, σ is length and $\phi(q_1, q_2) = 0$ at $|q_1 - q_2| = \sigma$. The phase diagram of the 2D Lennard-Jones system obtained from the equation of state by Reddy et al. [38] and ensemble simulation [39, 40] shows the vapor-liquid phase transition and the vapor-liquid coexistence. From Singh et al [39], the estimate of the critical temperature and density $T_c = 0.472$ and $\rho_c = 0.33 \pm 0.02$ are obtained.

2.2 Deep Learning and Neural Networks

In this section, some fundamental concepts in ML techniques including deep learning (DL) are explained. The types of machine learning are distinguished as supervised or unsupervised. Supervised learning is a method of learning from labeled data. In the unsupervised learning methods, the machine is concerned with finding patterns and structure in unlabeled data. We focus on supervised learning.

2.2.1 Feed-forward Neural Networks

Feed-forward neural networks, or multilayer perceptrons (MLPs), consist of units and are used for supervised learning. A multilayer perceptron in figure 2.1 is a perceptron with one or more hidden layers. Multilayer perceptron consists of three types of layers - input layer, hidden layer, and out layer. The multilayer perceptron multiplies the inputs by the weight for each input value, and the combined result value becomes the input value of the activation function. The activation functions help the network learn any complex relationship between input and output. After that, the result value of the activation function becomes the input value of the next node. In this way, the result of going through some hidden layers becomes the final output value. The training process of multilayer perceptrons learns the model in the direction of minimizing loss function by changing weights in the network. A type of loss function is one of the hyperparameters and needs to be deter-

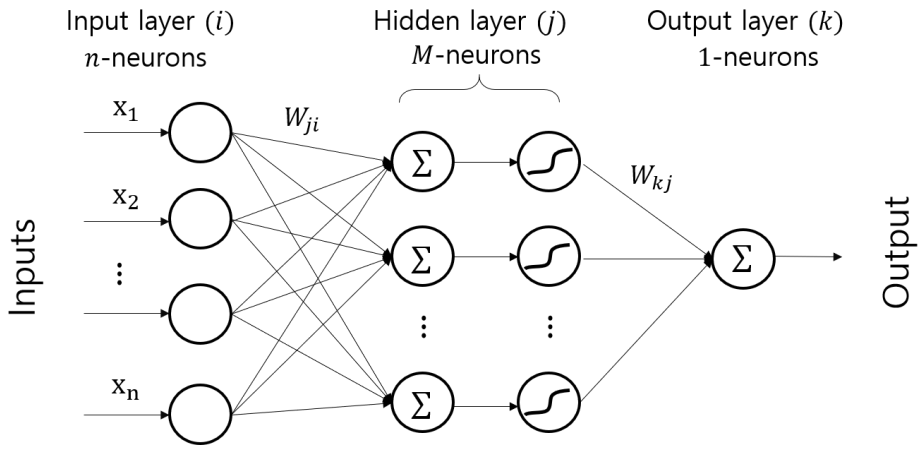

Figure 2.1: MLP neural networks

mined according to the given tasks. The loss function for regression is mean absolute error as follows:

$$\mathcal{L}_{MAE} = \frac{1}{N}\sum_{j}^{N}|y_j - \hat{y}_j| \tag{2.16}$$

,where $\hat{y}_j$ is the j-th value in the output as the prediction, y_j is the actual value. Whereas mean squared error is typically used as shown in Eq. (2.17).

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{j}^{N}(y_j - \hat{y}_j)^2 \tag{2.17}$$

The value of each weight is obtained in the reverse order of the method of calculating the final output, which is called backpropagation. The way to learn in the direction of minimizing the loss is gradient descent. The gradient descent in Eq. (2.18) is a way of taking small steps in the direction that decrease loss.

$$w \leftarrow w - \eta\frac{\partial Loss}{\partial w} \tag{2.18}$$

where w is for each learable parameter, η is the learning rate which controls the step size, and Loss stands for loss function and the difference between the actual value and the prediction.

2.2.2 Convolutional Neural Networks

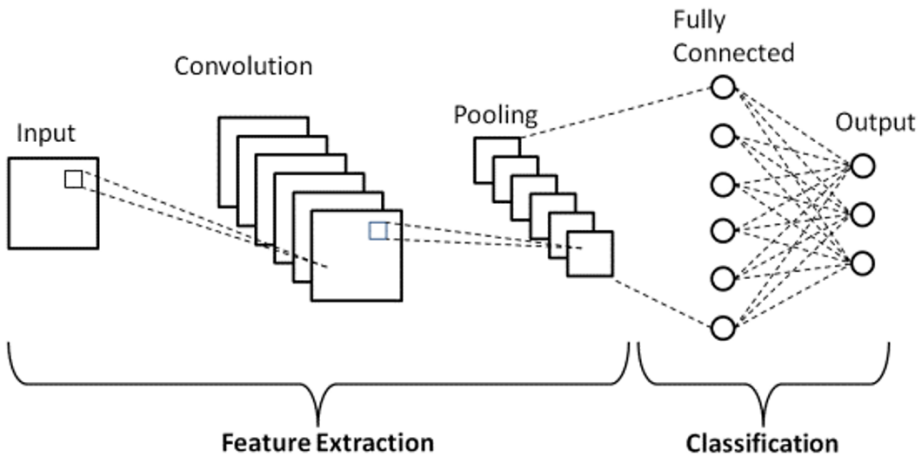

Figure 2.2: CNN architecture

Convolutional neural network (CNN) is known to use convolution operations to capture the spatial dependencies in an images, significantly detect the important features compared to feed-forward neural networks. The CNN are utilized for images classifications and images recognition. The CNN are divided into feature extraction in multiple hidden layers and classification in output layer as shown in figure 2.2. The feature extraction consists of convolution layers and pooling layers. One or multiple convolution layers extract the simple features from input by using convolution operations. The con-

volution layer applies filters to an input and generates the feature maps as shown in figure 2.3.

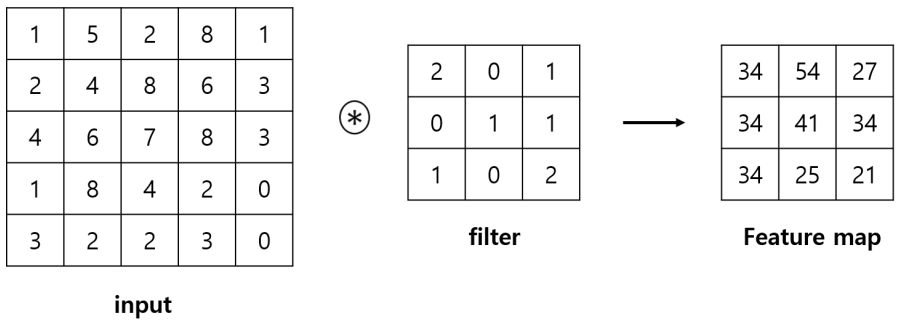

Figure 2.3: Element-wise multiplication of the input and filter. 3x3 convolution is performed on the 5x5 input. This convolution produces 3x3 feature map.

Following each convolution operation, the activation function is applied and add non-linearity into the network to learn complex structures in the data.

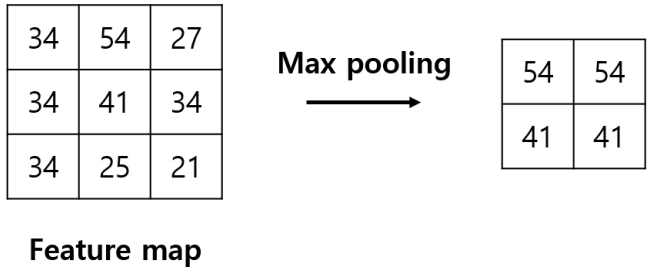

Figure 2.4: Max pooling using 2x2 filter with stride of 1 from the feature map. The maximum value of each filter is selected.

As shown in figure 2.4, the pooling layer after the convolution layer is used to reduce the dimensions of the feature maps but retain important information, and to also control overfitting . Once the features extracted by the convolution layers and pooling layers are created, the output from the flat-

ten layer that transforms into a 1D array of vector is fed to the fully connected layers for the classification. The final fully connected layer contains a softmax activation function and has the number of nodes as the number of classes. The softmax outputs a probability value from 0 to 1 for each of the classes for $j = 1...J$ given input z and add up to 1 in Eq. (2.19).

$$f_j(z) = \frac{e^{z_j}}{\sum_k^J e^{z_k}} \tag{2.19}$$

Chapter 3

Computational Details

In spin dynamics simulations and molecular dynamics simulations, the classical equations of motion are solved numerically using symplectic methods. This chapter describes these dynamics simulations tricks that were implemented.

3.1 Spin Dynamics Simulations

This section describes the symplectic method applied to Spin dynamics simulation.

3.1.1 Heisenberg Models

The ferromagnetic Heisenberg model on a cubic lattice as shown in Eq. (2.14) is used to demonstrate the efficiency of our method. We formalize our spin

dynamics following the notations of Tsai *et al.* [41]. The equations of motion for all spins are written as

$$\frac{d\sigma(t)}{dt} = \hat{R}\sigma(t), \tag{3.1}$$

where $\sigma(t) = (\boldsymbol{S}^1(t), \boldsymbol{S}^2(t), \ldots, \boldsymbol{S}^n(t))$ is the spin configuration at time t. The integration of the equations of motion in Eq. (3.1) is done using the second order Suzuki-Trotter decomposition method as in Tsai *et al.* [41].

The ferromagnetic Heisenberg model is considered on the cubic lattice of dimensions $L \times L \times L$ with periodic boundary conditions. In the spin dynamics approach, the equations of motion for the Heisenberg model is governed by the following equation:

$$\frac{d\boldsymbol{S}^i}{dt} = -\boldsymbol{S}^i \times \boldsymbol{H}^i_{\text{eff}} = \begin{bmatrix} 0 & -H^i_{\text{eff,z}} & H^i_{\text{eff,y}} \\ H^i_{\text{eff,z}} & 0 & -H^i_{\text{eff,x}} \\ -H^i_{\text{eff,y}} & H^i_{\text{eff,x}} & 0 \end{bmatrix} \boldsymbol{S}^i = R^i \boldsymbol{S}^i. \tag{3.2}$$

Here, $\boldsymbol{H}^i_{\text{eff}}$ is the effective field acting on the i^{th} spin. The k component of the effective field can be specified as $\boldsymbol{H}^i_{\text{eff,k}} = -\sum_{j=nn(i)} S^j_k$, where the sum runs over the nearest neighbor pairs of sites and $k = x, y$, and z.

3.1.2 Symplectic Algorithms

The symplectic methods are based on decompositions of exponential operators. As following the mathematical notations of Tsai et al., we decompose the evolution operator $\hat{R}$ into $\hat{R}_A$ and $\hat{R}_B$ on the sublattices A and B re-

spectively and obtain

$$e^{(\hat{R}_A+\hat{R}_B)\tau} = e^{\hat{R}_B\tau/2}e^{\hat{R}_A\tau}e^{\hat{R}_B\tau/2} + O(\tau^3). \tag{3.3}$$

The symplectic method determines temporal evolution of the spin orientations denoted as $\sigma(t) = \{\sigma_A(t), \sigma_B(t)\}$. The formal solution of the equations of motion of all spins can be written by using symplectic method as follows,

$$\sigma(t+\tau) \cong e^{\hat{R}_B\tau/2}e^{\hat{R}_A\tau}e^{\hat{R}_B\tau/2}\{\sigma_A(t), \sigma_B(t)\}. \tag{3.4}$$

For second-order Suzuki-Trotter decomposition method, the integration time step is limited up to $\tau \sim 0.04/J$ and for fourth-order Suzuki-Trotter decomposition method, the integration time step is limited up to $\tau \sim 0.2/J$ [41].

3.2 Classical Molecular Dynamics Simulations

The movement of particles is predicted by numerically solving the Hamiltonian equations of motion. This section describes the symplectic method applied to molecular dynamics simulation.

3.2.1 Hamiltonian System

A Hamiltonian system is a dynamical system described by the scalar function $H(\mathbf{q}_\alpha, \mathbf{p}_\alpha)$ of the phase space given by $(\mathbf{q}_\alpha, \mathbf{p}_\alpha) = (\mathbf{q}_{1,\alpha}, ..., \mathbf{q}_{n,\alpha}, \mathbf{p}_{1,\alpha}, ..., \mathbf{p}_{n,\alpha}) \in \Omega$, n is the total number of particles. In this paper for parallel computations in graphical processing units (GPU), we consider an ensemble of

independent hamiltonian systems index by α, $\alpha = 1, \cdots N$. The Hamilton's equations of motion with a Hamiltonian $H(\mathbf{q}_\alpha, \mathbf{p}_\alpha)$ as a conserved quantity are written as

$$\dot{\mathbf{q}}_\alpha = \frac{\partial H(\mathbf{q}_\alpha, \mathbf{p}_\alpha)}{\partial \mathbf{p}_\alpha}, \quad \dot{\mathbf{p}}_\alpha = -\frac{\partial H(\mathbf{q}_\alpha, \mathbf{p}_\alpha)}{\partial \mathbf{q}_\alpha}. \tag{3.5}$$

Consider particles with mass $m_{i,\alpha} = 1$ described by their position $\mathbf{q}_{i,\alpha}$ and momentum $\mathbf{p}_{i,\alpha}$ for i-th particle of α-th sample, interacting via a potential $\phi^{LJ}(|\mathbf{q}_{i,\alpha} - \mathbf{q}_{k,\alpha}|)$, $k \neq i$ is the index by the neighboring particle of the i-th particle and $m_{i,\alpha}$. The Hamiltonian function $H(\mathbf{q}_\alpha, \mathbf{p}_\alpha)$ of the system can be written as

$$H(\mathbf{q}_\alpha, \mathbf{p}_\alpha) = K(\mathbf{p}_\alpha) + U(\mathbf{q}_\alpha) = \sum_{i=1}^{n} \frac{\mathbf{p}_{i,\alpha}^2}{2m_{i,\alpha}} + \sum_{i,k,i\neq k} \phi^{LJ}(|\mathbf{q}_{i,\alpha} - \mathbf{q}_{k,\alpha}|). \tag{3.6}$$

The H is the total energy that is sum of kinetic and potential energies and is conserved with respect to time. We use Lennard-Jones system as a demonstration for physical model. All quantities are computed in dimensionless units.

3.2.2 Lennard-Jones Potential

For computation convenience and numerical stability, we want to rescale the system so that the box volume $|\Omega| = 1$. $U : \Omega^n \mapsto \mathbb{R}$. Eq. (2.15) can write q_1, q_2 in dimensionless units, $q_1 = \xi_1 L$, $q_2 = \xi_2 L$ where L is the box length. Then ξ is dimensionless. Break the LJ potential into ϕ_6 and ϕ_{12},

$$\phi(q_1, q_2) = 4\epsilon \left(\frac{\sigma^{12}}{L^{12}|\xi_1 - \xi_2|^{12}} - \frac{\sigma^6}{L^6|\xi_1 - \xi_2|^6} \right) \tag{3.7}$$

In dimensionless units, $\phi_6(\xi_1,\xi_2) = 1/|\xi_1-\xi_2|^6$, $\phi_{12}(\xi_1,\xi_2) = 1/|\xi_1-\xi_2|^{12}$. Then the relationship between dimensionless potential and original potential is,

$$\phi_6(q_1,q_2) = \frac{4\epsilon\sigma^6}{L^6}\frac{1}{|\xi_1-\xi_2|^6} = \frac{4\epsilon\sigma^6}{L^6}\phi_6(\xi_1,\xi_2) \tag{3.8}$$

$$\phi_{12}(q_1,q_2) = \frac{4\epsilon\sigma^{12}}{L^{12}}\frac{1}{|\xi_1-\xi_2|^{12}} = \frac{4\epsilon\sigma^{12}}{L^{12}}\phi_{12}(\xi_1,\xi_2) \tag{3.9}$$

We can also derive the derivative in w.r.t. the individual terms ϕ_6 and ϕ_{12}.

$$\Phi(q_1,q_2,\cdots q_n) = \sum_{i\neq k}\phi(q_i,q_k) \tag{3.10}$$

$$\Phi(q_1,q_2,\cdots q_n) = \frac{4\epsilon\sigma^{12}}{L^{12}}\sum_{i\neq k}\left(\frac{1}{|\xi_k-\xi_i|^{12}}\right) - \frac{4\epsilon\sigma^6}{L^6}\sum_{i\neq k}\left(\frac{1}{|\xi_k-\xi_i|^6}\right) \tag{3.11}$$

$$\frac{\partial}{\partial q_k} = \frac{\partial}{L\partial\xi_k} \tag{3.12}$$

$\vec{q}_k$ is position of k-particle which has two component $(q_{k,x}, q_{k,y})$. Write $\vec{q}_k = L\vec{\xi_k} = (L\xi_{k,x}, L\xi_{k,y})$ in dimensionless units.

$$\frac{\partial\Phi(q_1,q_2,\cdots q_n)}{\partial\vec{q}_k} = \left\{\frac{4\epsilon\sigma^{12}}{L^{13}}\sum_{i\neq k}\left(-12\frac{\xi_{k,x}-\xi_{i,x}}{(|\xi_k-\xi_i|)^{14}}\right) - \frac{4\epsilon\sigma^6}{L^7}\sum_{i\neq k}\left(-6\frac{\xi_{k,x}-\xi_{i,x}}{(|\xi_k-\xi_i|)^8}\right),\right.$$
$$\left.\frac{4\epsilon\sigma^{12}}{L^{13}}\sum_{i\neq k}\left(-12\frac{\xi_{k,y}-\xi_{i,y}}{(|\xi_k-\xi_i|)^{14}}\right) - \frac{4\epsilon\sigma^6}{L^7}\sum_{i\neq k}\left(-6\frac{\xi_{k,y}-\xi_{i,y}}{(|\xi_k-\xi_i|)^8}\right)\right\}$$

3.2.3 Initialization

The initial configurations are used to prepare the input of neural network model as the phase space $(\mathbf{q}_\alpha(0), \mathbf{p}_\alpha(0))$ that describes the initial state of system and prepared on gas+solid, gas+liquid, and gas regions at each temperature in two-dimensional phase diagram of Lennard-Jones systems. The dynamic properties at different temperatures with various densities are studied. Independent positions $\mathbf{q}_\alpha(0)$ are sampled using Monte-Carlo simulation with the Metropolis algorithm [36, 42, 43] for each temperature k_BT/J and consider the temperature dataset in the range of $k_BT/J \in [0.27 - 0.71]$. Initial momentum $\mathbf{p}_\alpha(0)$ are sampled using Boltzmann distribution at fixed temperature in the range of $k_BT/J \in [0.27 - 0.71]$.

3.2.4 Periodic Boundary Condition and Minimum Image Convention

Periodic boundary condition approximates a large systems by using small subsystem. The particles of the small subsystem are controlled in a simulation box. In the case that particle leaves the simulation box, identical particle of adjacent box enters the simulation box. The periodic boundary conditions use the minimum image convention for short ranged force and consider the interactions between particle and the closest image in the system.

3.2.5 Symplectic Algorithms

Hamilton's equation of motion can be solved using symplectic algorithms, for example, using the velocity-verlet algorithm in Eq. (3.13). The method performs the evolution of the configuration as $\mathbf{q}_{i,\alpha}(t)$ and $\mathbf{p}_{i,\alpha}(t)$ are updated to $\mathbf{q}_{i,\alpha}(t+\tau)$ and $\mathbf{p}_{i,\alpha}(t+\tau)$ for each i-th particle of α-th sample as

$$\mathbf{p}_{i,\alpha}(t+\frac{\tau}{2}) = \mathbf{p}_{i,\alpha}(t) + \dot{\mathbf{p}}_{i,\alpha}(t)\frac{\tau}{2} = \mathbf{p}_{i,\alpha}(t) + \left(- \left.\frac{dH}{d\mathbf{q}_{i,\alpha}}\right|_{t} \right)\frac{\tau}{2} \tag{3.13}$$

$$\mathbf{q}_{i,\alpha}(t+\tau) = \mathbf{q}_{i,\alpha}(t) + \mathbf{p}_{i,\alpha}(t+\frac{\tau}{2})\tau$$

$$\mathbf{p}_{i,\alpha}(t+\tau) = \mathbf{p}_{i,\alpha}(t+\frac{\tau}{2}) + \dot{\mathbf{p}}_{i,\alpha}(t+\tau)\frac{\tau}{2} = \mathbf{p}_{i,\alpha}(t+\frac{\tau}{2}) + \left(- \left.\frac{dH}{d\mathbf{q}_{i,\alpha}}\right|_{t+\tau} \right)\frac{\tau}{2}$$

where τ is the integration time step.

Chapter 4

Deep Learning Approach to Spin Dynamics

This chapter introduces the Deep Learning method how to make corrections to achieve higher accuracy in spin dynamics and validate the performance of the Deep Learning method.

4.1 Supervised Deep Learning Method

A fully supervised Deep Learning method is developed to perform the spin dynamics by using the second order Suzuki-Trotter decomposition method to reduce simulation errors.

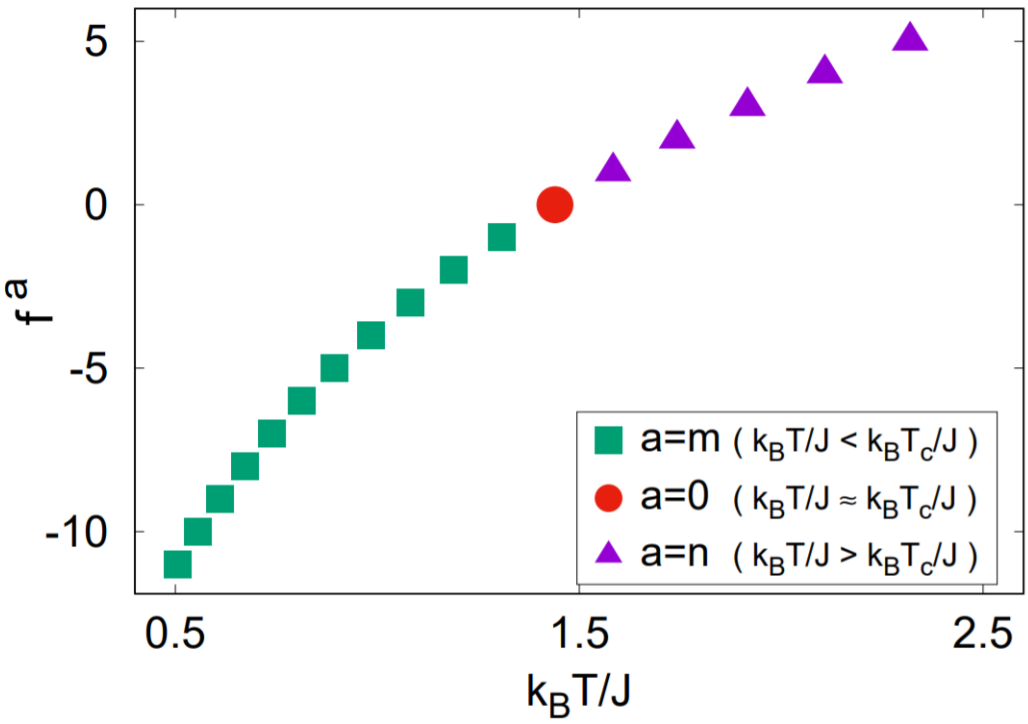

Figure 4.1: Plot of f^{a} as a function of temperature k_BT/J used for the simulated annealing methods, where f is modification factor and a is power of f.

4.1.1 Data Preparation

In order to produce training data for our supervised Deep Learning, initial spin configurations are considered at ordered, near-critical, and disordered states in the temperature range $k_BT/J \in [0.5, 2.4]$ and sampling 9.1×10^5 independent spin configurations using Monte-Carlo simulations with the Met ropolis–Hastings algorithm [36, 42, 43]. The initial spin configurations are prepared with 300,000 samples in ordered states, 210,000 samples near critical states, and 400,000 samples disordered states by simulated annealing

method. In the Monte Carlo simulation, transition probability from one state to another state is $W = e^{-\beta\Delta E}$ for $\Delta E < 0$ in the metropolis algorithm. Since an inverse temperature β goes to infinite at low temperature, it is easy to fall into the local minima because the transition probability is $W = 0$. For the purpose of avoiding the local minima problem, we gradually lower the temperature from a high temperature to a low temperature using the simulated annealing method. Temperature dataset shown in figure 4.1 are generated in range of temperature $k_BT/J \in [0.5, 2.4]$, where k_BT/J is defined as follows:

$$\begin{aligned} k_BT/J &= f^{-m} k_BT_c/J \quad , \; k_BT/J < k_BT_c/J \\ k_BT/J &= f^{n} k_BT_c/J \quad , \; k_BT/J > k_BT_c/J \; . \end{aligned} \tag{4.1}$$

Here, the positive real number f, the modification factor, is used to adjust the number of temperature dataset between minimum temperature $k_BT_{min}/J = 0.5$ and maximum temperature $k_BT_{max}/J = 2.4$, where m is the number of temperature dataset between the critical temperature k_BT_c/J and k_BT_{max}/J and n is the number of temperature dataset between k_BT_{min}/J and k_BT_c/J. The values of m and n are estimated as follows:

$$\begin{aligned} m &= \frac{log k_BT_c/J - log k_BT/J}{log f} \\ n &= \frac{log k_BT/J - log k_BT_c/J}{log f} \; , \end{aligned} \tag{4.2}$$

where $k_BT_c/J \approx 1.44$, $m = 11$, $n = 5$, and $f = 1.1$ are used in this paper. The initial spin configurations are obtained below, near, and above the critical temperature. Below the critical temperature, 30,000 spin configurations

representing order states are generated by using the Monte Carlo simulation at temperatures $k_BT/J = 0.5$, 0.56, 0.61, 0.67, 0.74, 0.81, 0.89, 0.98, 1.08, 1.19 each. Near the critical temperature, 70,000 spin configurations are generated at temperatures $k_BT/J = 1.31$, 1.44, 1.58 each. Above the critical temperature, 100,000 spin configurations representing disordered states are generated at temperatures $k_BT/J = 1.74$, 1.92, 2.11, 2.32 each.

Below the critical temperature, the entropy is low so the generated number of initial spin configurations are smaller than other temperatures. As the temperature increases as the entropy is high, we increase the number of initial spin configurations.

The temperatures for annealing are gradually lowered from high to low temperatures and Monte Carlo data are always obtained at equilibrium configurations. For each sampled initial spin configuration σ_i, two sets of spin dynamics simulations are performed with the time steps $\tau_1 = 10^{-1}$ and $\tau_3 = 10^{-3}$ as illustrated in figure 4.2. Second-order Suzuki-Trotter method uses $\tau = 0.04$ as typical integration time step, so we use $\tau = 10^{-3}$ which would give good accurate simulation. For large time step, we tried $\tau = 10^{-2}$ and $\tau = 10^{-1}$, with our Deep Learning corrections, a large time step of $\tau = 10^{-1}$ gives the best speed up with a good accuracy. The spin configuration with time step $\tau_3 = 10^{-3}$ needs 100 time steps of simulations to pair with the spin configuration with one time step $\tau_1 = 10^{-1}$. Formally, we represent the updated spin configurations $\sigma_i^{(10^{-1})}$ and $\sigma_i^{(10^{-3})}$ by using the Suzuki-Trotter

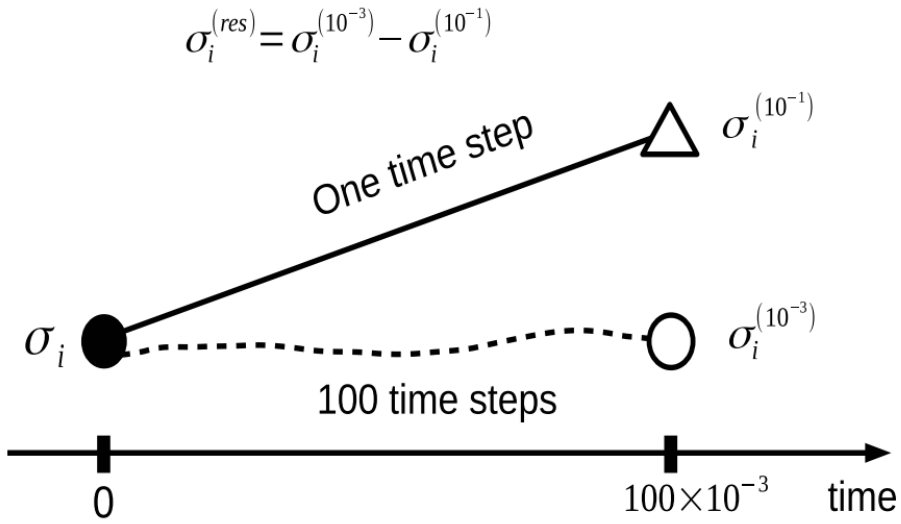

Figure 4.2: Spin configurations for training data preparation. σ_i is initial spin configuration, $\sigma_i^{(10^{-1})}$ is spin configuration after one time step of $\tau_1 = 10^{-1}$ from σ_i, and $\sigma_i^{(10^{-3})}$ is spin configuration after 100 time steps of $\tau_3 = 10^{-3}$ from σ_i. $\sigma_i^{(res)}$ is residue of $\sigma_i^{(10^{-3})}$ and $\sigma_i^{(10^{-1})}$.

position method as

$$\begin{aligned} \sigma_i^{(10^{-1})} &\leftarrow \mathrm{e}^{\hat{R}_B\tau_1/2}\mathrm{e}^{\hat{R}_A\tau_1}\mathrm{e}^{\hat{R}_B\tau_1/2}\sigma_i, \quad \tau_1 = 10^{-1} \\ \sigma_i^{(10^{-3})} &\leftarrow (\mathrm{e}^{\hat{R}_B\tau_3/2}\mathrm{e}^{\hat{R}_A\tau_3}\mathrm{e}^{\hat{R}_B\tau_3/2})^{100}\sigma_i, \quad \tau_3 = 10^{-3} \qquad i = 1, ...D, \end{aligned} \tag{4.3}$$

where σ_i is an initial spin configuration and D represents the number of training data. The difference between spin configuration $\sigma_i^{(10^{-3})}$ generated using $\tau_3 = 10^{-3}$ and spin configuration $\sigma_i^{(10^{-1})}$ generated using $\tau_1 = 10^{-1}$ is captured by

$$\sigma_i^{(res)} = \sigma_i^{(10^{-3})} - \sigma_i^{(10^{-1})} \qquad i = 1, \ldots, D \ , \tag{4.4}$$

where $\sigma_i^{(res)}$ is residue.

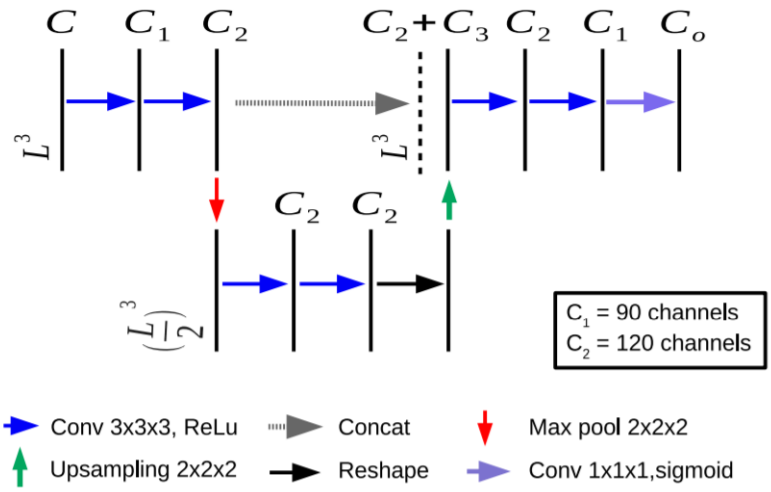

Figure 4.3: Illustration of the U-Net architecture. The architecture consists of encoder and decoder layers. Each vertical black line represents a multi-channel feature map. The number of channels is denoted on the top of straight vertical black line and each map's dimension is indicated on the left edge. Vertical dashed black lines correspond on the copied feature maps from each encoder layer.

4.1.2 U-Net Architecture

For our Deep Learning, initial spin configuration σ_i and spin configuration $\sigma_i^{(10^{-1})}$ are used as the inputs into U-Net [44], a kind of convolutional neural networks. The U-Net is a proven architecture for image segmentation as well as for extracting subtle features. The architecture of U-Net described in figure 4.3 is used for $4 \times 4 \times 4$ cubic lattice. Convolutional layers are used as an encoder followed by a decoder that consists of upsamplings and concatenations with the correspondingly feature maps from the encoder. The input dimensions of U-Net are reshaped to $[D, L, L, L, C]$ as cubic grid vector map. D is total number of training data and input channels C is 6 (3+3) by con-

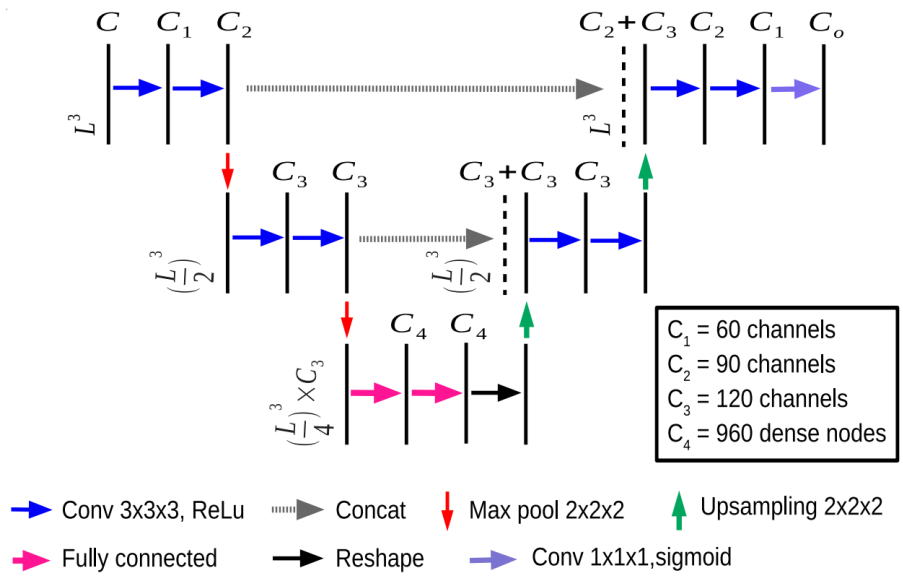

Figure 4.4: Illustration of the U-Net architecture. Each vertical black line represents a multi-channel feature map. The number of channels is denoted on the top of the straight vertical black line and each map's dimension is indicated on the left edge. Vertical dashed black lines correspond on the copied feature maps from each encoder layer.

catenating spin coordinates S_x, S_y, and S_z of σ_i and $\sigma_i^{(10^{-1})}$, respectively. The encoder consists of the repeated two convolutional layers with $3 \times 3 \times 3$ filters followed by a $2 \times 2 \times 2$ max pooling. Every step in decoder consists of upsampling layers with a $2 \times 2 \times 2$ filters followed by the repeated two convolutional layers with $3 \times 3 \times 3$ filters and copy with correspondingly cropped feature map from encoding layers. The periodic boundary conditions are also applied to the convolutional layers. The activation function of the output is a sigmoid for predicting values of residue with $[D, L, L, L, C_o]$ dimensions, where the number of output channels C_o is 3.

The architecture of U-net used for $8 \times 8 \times 8$ cubic lattice as shown in figure 4.4 is that convolutional layers are used as an encoder on left upper side followed by a decoder on right upper side that consists of upsamplings and concatenations with the correspondingly feature maps from the encoder. We add fully connected layers (FC) in the bottom of the network between the encoder and the decoder to efficiently determine particular weights in the feature map from the encoder , such as capturing more information of spin-spin interactions. The input channels C are 6 by concatenating spin coordinates S_x, S_y, and S_z of both σ_i and $\sigma_i^{(10^{-1})}$, respectively. The input dimensions of U-Net are reshaped to $[D, L, L, L, C]$ as cubic grid vector map, where D is the total number of training data, L is lattice size, and C is input channels. The encoder consists of the repeated two convolutional layers with $3\times3\times3$ filters followed by a $2\times2\times2$ max pooling. We apply a reshaping function to FC with dimensions from $[D, \frac{L}{4} \times \frac{L}{4} \times \frac{L}{4} \times C_4]$ into $[D, \frac{L}{4}, \frac{L}{4}, \frac{L}{4}, C_4]$. Every step in decoder consists of upsampling layers with a $2\times2\times2$ filters followed by the repeated two convolutional layers with $3\times3\times3$ filters and copies with correspondingly cropped feature maps from encoding layers. The periodic boundary conditions are also applied to the convolutional layers. The activation function of the output is a sigmoid for predicting values of residue with $[D, L, L, L, C_o]$ dimensions, where the number of output channels C_o is 3.

4.1.3 Deployment of U-Net

To deploy the trained U-Net for spin dynamics, spin dynamics simulation is carried out with one large time step $\tau_1 = 10^{-1}$ and this simulation result $\sigma_i^{(10^{-1})}$ can be used to predict $\sigma_i^{(10^{-3})}$ as follows:

$$\hat{\sigma}_i^{(10^{-3})} = \sigma_i^{(10^{-1})} + \hat{\sigma}_i^{(res)} \simeq \sigma_i^{(10^{-3})}, \tag{4.5}$$

where $\hat{\sigma}_i^{(10^{-3})}$ is the predicted spin configuration for 100 time steps of $\tau_3 = 10^{-3}$ and predicted residue $\hat{\sigma}_i^{(res)}$ is the correction term by Deep Learning. A sequence of spin dynamics is conducted at $\tau_1 = 10^{-1}$ and for each step, Eq. (4.5) is used to perform corrections as shown in figure 4.5. This new time integration scheme is repeated up to maximum time t_{max}. This scheme requires only forward propagation using the GPU implemented with TensorFlow library [45], so the computing time is negligible.

4.1.4 Normalization of Residue

The difference between spin configuration generated with $\tau_3 = 10^{-3}$ and that generated with $\tau_1 = 10^{-1}$ is captured by residue $\sigma_i^{(res)}$ as shown in Eq. (4.4). Let $(\sigma_i^{(res)})_k^j$ be the k component of residual spin at site j of the lattice, and k denotes x, y, and z components. The values of $(\sigma_i^{(res)})_k^j$ can be quite small for some simulations, to maintain numerical stability, we normalize these values as follows. Each component $(\sigma_i^{(res)})_k^j$ over D samples of training data is normalized to a range of [0,1] by fitting to have a Gaussian distribution, and

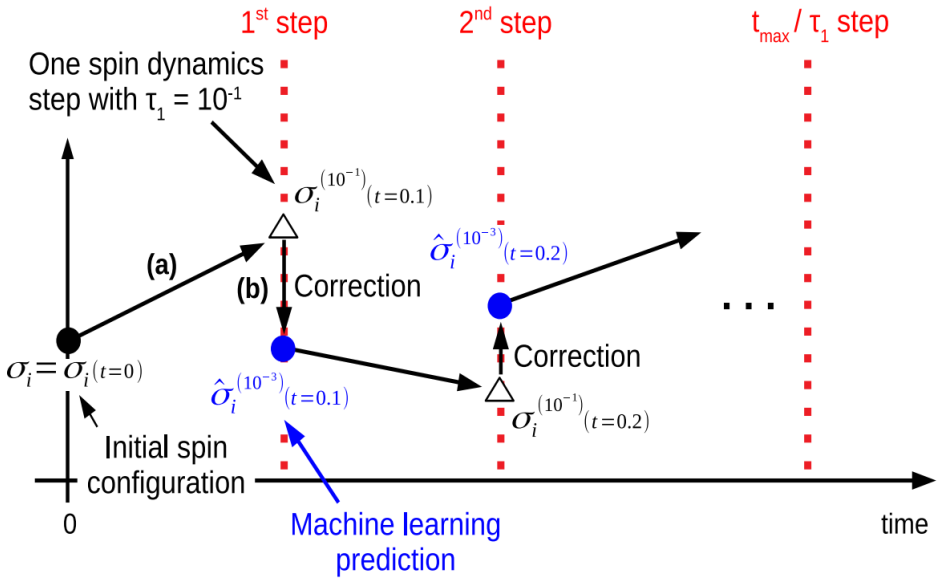

Figure 4.5: A sequence of spin dynamics for testing the trained U-Net model: (a) conduct one time step $\tau_1 = 10^{-1}$ of spin dynamics simulation; (b) use $\sigma_i^{(10^{-1})}$ to predict the spin configuration $\sigma_i^{(10^{-3})}$ by estimating predicted residue $\hat{\sigma}_i^{(res)}$ using Eq. (4.5). Steps (a) and (b) are repeated up to t_{max} time.

find the mean and standard deviation for each k component, respectively. For lattice size $L = 4$, $\lambda_{min} = -0.22455$ and $\lambda_{max} = 0.22455$ are defined by taking 11 times the largest standard deviation of k component. 11 standard deviations translates to a p-value of 1.911×10^{-28}, which ensures that during inference, the normalized residue $(\sigma_i^{(res)})_k^j$ is always within the range [0,1]. For lattice size $L = 8$, $\lambda_{min} = -0.25472$ and $\lambda_{max} = 0.25472$ are defined by taking 13 times the largest standard deviation of k component. Finally, each component $(\sigma_i^{(res)})_k^j$ is normalized to the range $[0, 1]$ and guarantee sta-

ble convergence of weights and biases in Deep Learning as follows :

$$(\sigma_i^{norm})_k^j = \frac{(\sigma_i^{(res)})_k^j - \lambda_{min}}{\lambda_{max} - \lambda_{min}} \qquad (k = x, y, z,\ \ i = 1, ...D). \tag{4.6}$$

During the prediction, $(\sigma_i^{(res)})_k^j$ from test data is normalized to a range of [0, 1] by using λ_{min} and λ_{max}, which have already been obtained.

Loss Function and Training

The loss function for one data point of $(\sigma_i, \sigma_i^{(10^{-1})}, \sigma_i^{norm})$ is the mean-square error between the normalized residue σ_i^{norm} and the predicted normalized residue $\hat{\sigma}_i^{norm}$ and is defined as

$$\mathcal{L}(\sigma_i, \sigma_i^{(10^{-1})}, \sigma_i^{norm}) = \frac{1}{L^3}\sum_{j=1}^{L^3} (\sigma_i^{norm})^j - (\hat{\sigma}_i^{norm})^j{}_2^2 \ , \tag{4.7}$$

where j is the index of lattice sites. The distance function between the j^{th} site of σ_i^{norm} and the j^{th} site of $\hat{\sigma}_i^{norm}$ is the sum of the square difference of all spin components :

$$(\sigma_i^{norm})^j - (\hat{\sigma}_i^{norm})^j{}_2^2 = \sum_{k=x,y,z} \left((\sigma_i^{norm})_k^j - (\hat{\sigma}_i^{norm})_k^j\right)^2 \ , \tag{4.8}$$

where i is the index of training data.

4.1.5 Converting $\hat{\sigma}_i^{norm}$ to $\hat{\sigma}_i^{(res)}$

For our Deep Learning, inputs into U-Net are obtained initial spin configurations σ_i and spin configurations $\sigma_i^{(10^{-1})}$ generated by spin dynamics simulations, and output is $\hat{\sigma}_i^{norm}$. We finally predict the spin configuration for

100 time steps of $\tau_3 = 10^{-3}$ using trained Deep Learning model as $\hat{\sigma}_i^{(10^{-3})} = \sigma_i^{(10^{-1})} + \hat{\sigma}_i^{(res)}$, where the predicted residue $\hat{\sigma}_i^{(res)}$ can be obtained by the following converting formula as $\hat{\sigma}_i^{(res)} = \hat{\sigma}_i^{norm}(\lambda_{max} - \lambda_{min}) + \lambda_{min}$.

4.2 Results and Discussions

The effectiveness of our proposed Deep Learning method is evaluated at $k_BT/J = 0.4 < k_BT_c/J$, $k_BT/J = 1.44 \approx k_BT_c/J$, and $k_BT/J = 2.4 > k_BT_c/J$. Note that at $k_BT/J = 2.4$, the system is in a disordered state and spatial corrections between spins are very short. One hundred independent spin configurations are generated by using Monte-Carlo simulation for use as test data sets at each temperature $k_BT/J = 0.4$, 1.44, and 2.4. Second order Suzuki-Trotter decomposition methods are used for all experiments.

4.2.1 Spin-Spin Correlation using Reference Trajectory

To evaluate the accuracy of simulation results, correlation is investigated by comparing spin dynamics trajectory $\sigma(t)$ with highly accurate spin dynamics trajectory $\rho(t)$ performed with $\tau = 10^{-6}$. $\tau = 10^{-6}$ is used as the reference time step as we found that it can give accurate trajectories. Correlation $\xi(t)$ as function of time t in which $\sigma(t)$ and $\rho(t)$ are compared is given by

$$\xi(\sigma, t) = \frac{1}{L^3}\sum_{j=1}^{L^3}[(\rho^j(t))_x(\sigma^j(t))_x + (\rho^j(t))_y(\sigma^j(t))_y + (\rho^j(t))_z(\sigma^j(t))_z], \quad (4.9)$$

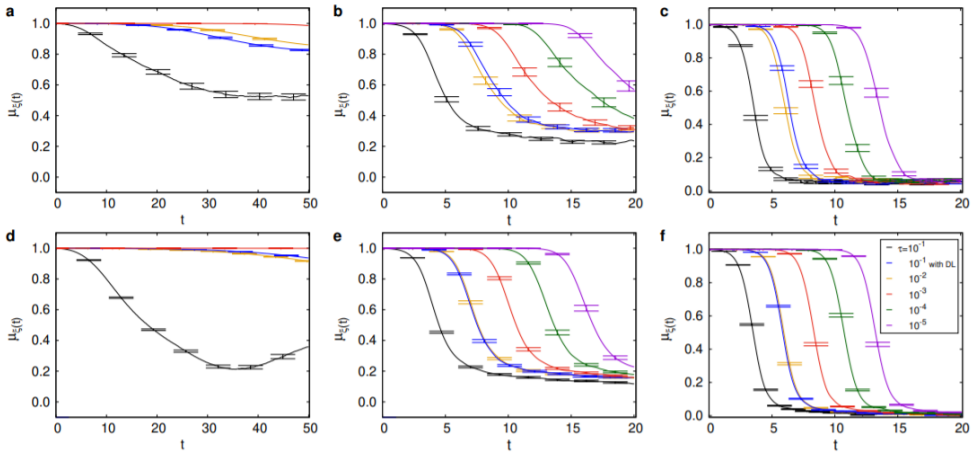

Figure 4.6: Spin-spin correlation using reference trajectory generated at $\tau = 10^{-6}$. Analysis of the mean of correlation $\mu_{\xi(t)}$ as a function of time on $4 \times 4 \times 4$ cubic lattice at $\mathbf{a}, k_BT/J = 0.4$, $\mathbf{b}, k_BT_c/J \approx 1.44$, and $\mathbf{c}, k_BT/J = 2.4$ and those on $8 \times 8 \times 8$ cubic lattice at $\mathbf{d}, k_BT/J = 0.4$, $\mathbf{e}, k_BT_c/J \approx 1.44$, and $\mathbf{f}, k_BT/J = 2.4$. Blue line presents the Deep Learning result while black line, yellow line, and red line are the simulation results for $\tau = 10^{-1}$, $\tau = 10^{-2}$, and $\tau = 10^{-3}$, respectively. Especially, at $k_BT/J = 1.44$ and $k_BT/J = 2.4$, green line and violet line show the simulation results for $\tau = 10^{-4}$ and $\tau = 10^{-5}$, respectively.

where index j denotes lattice site of spins, L is the linear dimension of the lattice, and L^3 is total number of spins at lattice sites. Since the initial spin configurations are the same, $\rho(0)$ is identical to $\sigma(0)$. We compute one hundred correlation $\xi(\sigma_i, t)$ for spin configurations $\sigma_i(t)$, where i is from 1 to 100. Then, we also estimate the mean of correlation $\mu_{\xi(t)}$ and the standard deviation of correlation std$(\xi(t))$ of $\xi(\sigma_i, t)$ as a function of time at each temperature. In figure 4.6, the spin-spin correlation plots are shown as using reference trajectory generated at the reference time step $\tau = 10^{-6}$ for $k_BT/J = 0.4$ ($k_BT/J < k_BT_c/J$) [figure 4.6a and 4.6d], $k_BT/J =$

1.44 ($k_BT/J \approx k_BT_c/J$) [figure 4.6b and 4.6e], and $k_BT/J = 2.4$ ($k_BT/J > k_BT_c/J$) [figure 4.6c and 4.6f]. At $k_BT/J < k_BT_c/J$, correlations remain high (red line, yellow line, and blue line) except for at $\tau = 10^{-1}$ without Deep Learning corrections (black line), where correlation drops around $t = 2$. This is due to accumulation of errors for large time steps. Correlation is recovered with Deep Learning corrections (blue line). Indeed correlations of $\tau = 10^{-1}$ with Deep Learning corrections are as good as for $\tau = 10^{-2}$ without Deep Learning corrections (yellow line), demonstrating a ~ 10 times speed up. At $k_BT/J \approx k_BT_c/J$ and $k_BT/J > k_BT_c/J$, spin-spin correlation drops faster than $k_BT/J < k_BT_c/J$ even for short time steps, $\tau = 10^{-4}$ (green line) and $\tau = 10^{-5}$ (violet line), due to disorder in the spin lattices. We define threshold time t_{thres} as the average time required for spin-spin correlation $\mu_{\xi(t)}$ to drop from 1 to 0.99. In figure 4.7, the plot of t_{thres} as a function of temperature k_BT/J has the logarithmic scale on the y-axis, and simulations for $\tau = 10^{-3}$ have higher threshold time (red squares) at each temperature than for $\tau = 10^{-1}$ without Deep Learning corrections. Threshold time (filled blue diamonds) for $\tau = 10^{-1}$ with Deep Learning corrections approaches to almost the same threshold time (yellow circles) for $\tau = 10^{-2}$ without Deep Learning corrections at each temperature.

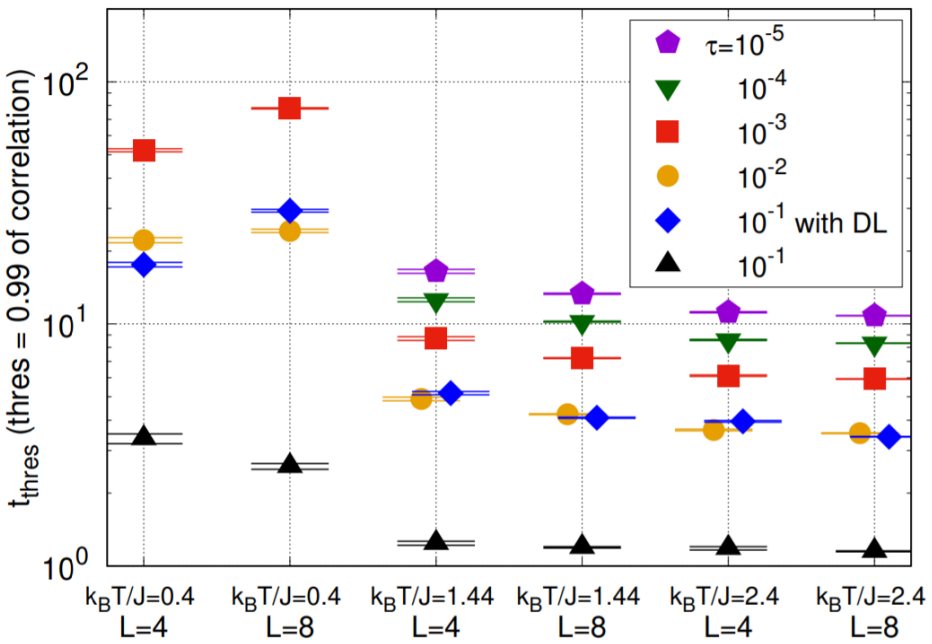

Figure 4.7: Threshold time t_{thres} as function of temperature. Filled blue rhombus represents the Deep Learning result while filled black triangles, filled yellow circles, filled red squares, filled green inverted triangles, and filled violet pentagons are the simulation results without DL corrections for $\tau = 10^{-1}$, $\tau = 10^{-2}$, $\tau = 10^{-3}$, $\tau = 10^{-4}$, and $\tau = 10^{-5}$, respectively.

4.2.2 Conservation of Energy and Magnetization

Suzuki-Trotter decomposition method provides important properties such as conservation of energy $\boldsymbol{e} = -L^{-3}\sum_{<i,j>}^{L^3} \boldsymbol{S}^i \cdot \boldsymbol{S}^j$ and magnetization $\boldsymbol{m} = L^{-3}\sqrt{\left(\sum_i S_x^i\right)^2 + \left(\sum_i S_y^i\right)^2 + \left(\sum_i S_z^i\right)^2}$, and time reversibility. We wish to compare the conservation of energy and magnetization across one hundred samples, but their starting spin configurations are different. In order to take statistics across the samples, we shift the energy and magnetization of the initial spin configurations to zero. Eq. (4.10) show how we shift the energy

per site $e(t)$ and magnetization per site $m(t)$ at each time step t. Here, Q represents the number of samples at each temperature. We use Q as one hundred.

$$\begin{aligned} \tilde{e}_i(t) &= e_i(t) - e_i(0) \qquad i = 1, \ldots, Q \\ \tilde{m}_i(t) &= m_i(t) - m_i(0) \qquad i = 1, \ldots, Q \ . \end{aligned} \tag{4.10}$$

With the shifting of energy and magnetization, we can compute the mean of absolute energy per site $\mu_{|\tilde{e}(t)|}$, the mean of absolute magnetization per site $\mu_{|\tilde{m}(t)|}$, standard deviation of energy per site std($\tilde{e}(t)$), and standard deviation of magnetization per site std($\tilde{m}(t)$) over independent samples.

Figure 4.8 ($L = 4$) and figure 4.9 ($L = 8$) show $\mu_{|\tilde{e}(t)|}$, std($\tilde{e}(t)$), $\mu_{|\tilde{m}(t)|}$, and std($\tilde{m}(t)$) as a function of time at $k_BT/J = 0.4$ ($k_BT/J < k_BT_c/J$) [figure 4.8a and 4.9a], $k_BT/J = 1.44$ ($k_BT/J \approx k_BT_c/J$) [figure 4.8b and 4.9b], and $k_BT/J = 2.4$ ($k_BT/J > k_BT_c/J$) [figure 4.8c and 4.9c]. For time steps $\tau = 10^{-2}$ (yellow line) and $\tau = 10^{-3}$ (red line), conservation of both energy and magnetization is good, as shown by the relatively constant mean plots ($\mu_{|\tilde{e}(t)|}$ and $\mu_{|\tilde{m}(t)|}$) and small standard deviations (std($\tilde{e}(t)$) and std($\tilde{m}(t)$)) across independent simulations. At $k_BT/J < k_BT_c/J$ and $k_BT/J \approx k_BT_c/J$, both energy and magnetization are not conserved in simulations without Deep Learning corrections for time step $\tau = 10^{-1}$ (black line). On the other hand, conservation is recovered using Deep Learning corrections (blue line). In figure 4.8c, at $k_BT/J > k_BT_c/J$, the system is disordered and the mean of absolute energy $\mu_{|\tilde{e}(t)|}$ and the mean of absolute magnetization $\mu_{|\tilde{m}(t)|}$ be-

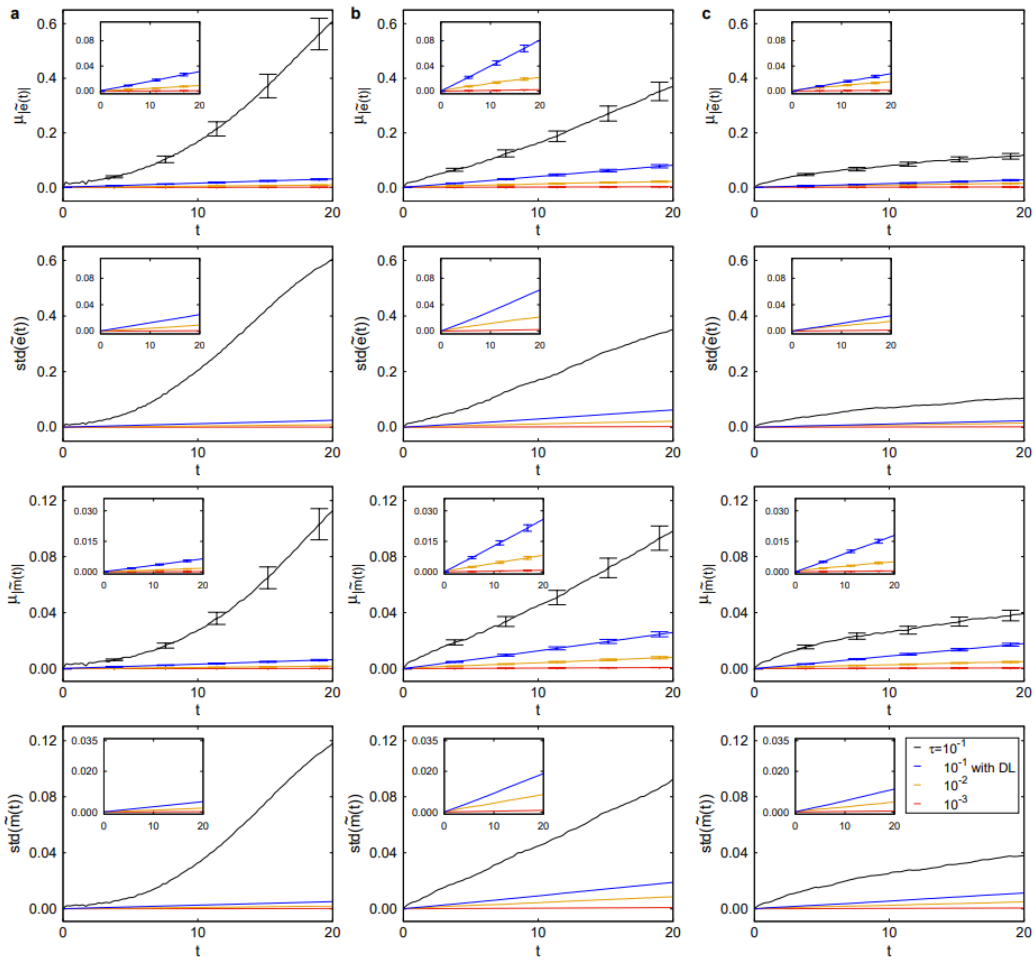

Figure 4.8: Conservation of energy and magnetization on $4 \times 4 \times 4$ cubic lattice. Predictions of the mean of absolute energy per site $\mu_{|\tilde{e}(t)|}$, standard deviation of energy per site $\mathrm{std}(\tilde{e}(t))$, the mean of absolute magnetization per site $\mu_{|\tilde{m}(t)|}$, and standard deviation of magnetization per site $\mathrm{std}(\tilde{m}(t))$ as a function of time at $\mathbf{a}, k_BT/J = 0.4$, $\mathbf{b}, k_BT_c/J \approx 1.44$, and $\mathbf{c}, k_BT/J = 2.4$. Black line, yellow line, and red line represent data obtained from spin dynamics simulations with $\tau = 10^{-1}$, $\tau = 10^{-2}$, and $\tau = 10^{-3}$, respectively, while blue line represents data from Deep Learning correction.

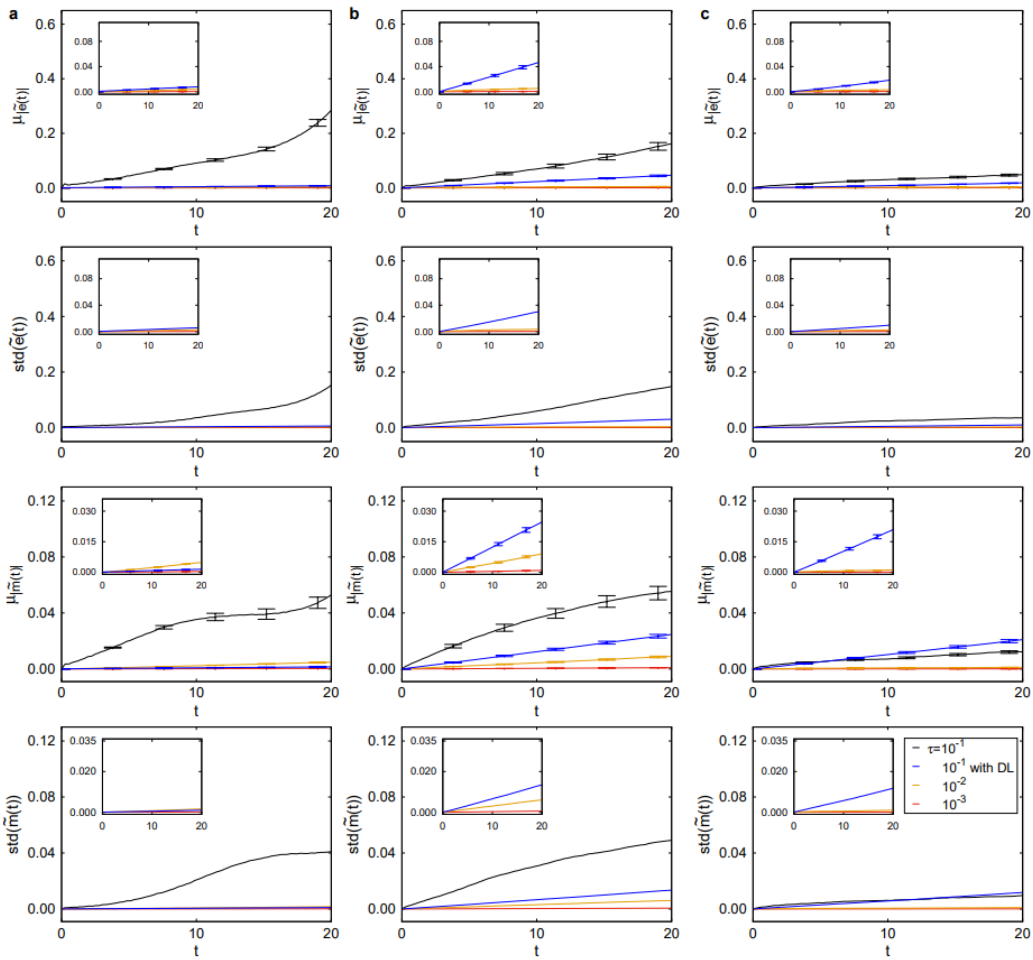

Figure 4.9: Conservation of energy and magnetization on $8 \times 8 \times 8$ cubic lattice. Predictions of $\mu_{|\tilde{e}(t)|}$, $\mathrm{std}(\tilde{e}(t))$, $\mu_{|\tilde{m}(t)|}$, and $\mathrm{std}(\tilde{m}(t))$ as a function of time at **a**, $k_BT/J = 0.4$, **b**, $k_BT_c/J \approx 1.44$, and **c**, $k_BT/J = 2.4$. Black line, yellow line, and red line represent data obtained from spin dynamics simulations with $\tau = 10^{-1}$, $\tau = 10^{-2}$, and $\tau = 10^{-3}$, respectively, while blue line represents data from Deep Learning correction. These figures show that the effect of averaging over disordered spins for $L = 8$ is stronger than for $L = 4$ above the critical temperature k_BT_c/J.

come more constant, simply due to averaging of disordered spins. Especially, figure 4.9c shows that at $k_BT/J > k_BT_c/J$, the effect of averaging over disordered spins for $L = 8$ is stronger than for $L = 4$. At high temperature, the number of possible states increase exponentially and hence fitting by Deep Learning corrections is more difficult.

Chapter 5

Deep Learning Approach to Molecular Dynamics

This chapter introduces the Deep Learning method used to learn effective force to recover simulation stability and validate the performance of the Deep Learning method.

5.1 Supervised Deep Learning Method

5.1.1 Data Preparations

Let us consider a system of the 2-dimensional Lennard-Jones small n=16 particles system. The density of the system is chosen to be ρ = 0.1, 0.14, 0.2, 0.27, and 0.38 and the particles are confined to a square box of length

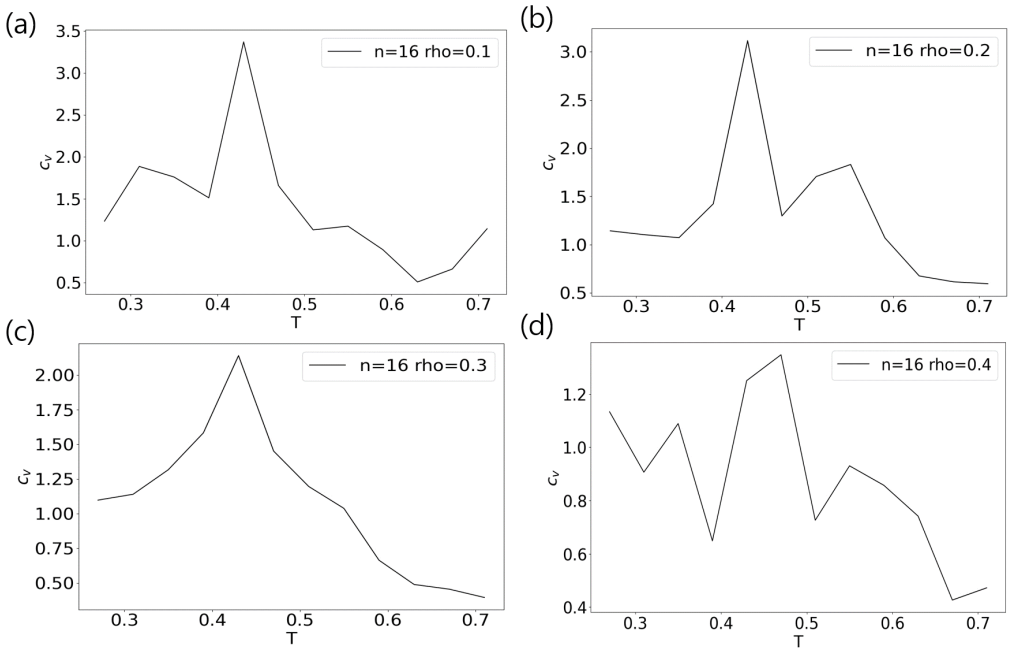

Figure 5.1: Specific heat c_v in two-dimensional LJ potential on N=16 particles at the range of $k_BT/J \in [0.27 - 0.71]$ that is critical temperature $T_c = 0.472$ with (a) $\rho = 0.1$, (b) $\rho = 0.2$, (c) $\rho = 0.3$ and, (d) $\rho = 0.4$.

$L = \sqrt{n/\rho}$. The initial configurations are used to prepare the input of neural network model as a set of coordinates $(\mathbf{q}(0), \mathbf{p}(0))$ that describe the initial state of system and total sampling 1,200,000 independent configurations with a various densities ρ. Independent $\mathbf{q}(0)$ are sampled using Monte-Carlo simulation with the Metropolis algorithm for each temperature k_BT/J and consider the temperature dataset in the range of $k_BT/J \in [0.27 - 0.71]$. From Singh et al. [39], the estimate of the critical temperature and density $T_c = 0.472$ and $\rho_c = 0.33 \pm 0.02$ are obtained. Fig 5.1 shows the high specific heat per particle near critical temperature T_c about one sample in a Monte-

Carlo simulation. Initial positions $\mathbf{q}_\alpha(0)$ are prepared with $60,000$ samples at the range of $k_BT/J \in [0.27, 0.39]$ on gas+solid regions, $60,000$ samples at the range of $k_BT/J \in [0.43, 0.55]$ on gas+liquid regions and, $120,000$ samples at the range of $k_BT/J \in [0.59, 0.71]$ on gas regions with each density in two-dimensional phase diagram of Lennard-Jones systems as shown in figure 5.2.

n	ρ	L	Phase	T	# of (q(0), p(0))
16	0.1	12.65	gas+solid	0.27, 0.31, 0.35, 0.39	15,000x4 = 60,000
			gas+liquid	0.43, 0.47, 0.51, 0.55	15,000x4 = 60,000
			gas	0.59, 0.63, 0.67, 0.71	30,000x4 = 120,000
	0.14	10.69	gas+solid	0.27, 0.31, 0.35, 0.39	15,000x4 = 60,000
			gas+liquid	0.43, 0.47, 0.51, 0.55	15,000x4 = 60,000
			gas	0.59, 0.63, 0.67, 0.71	30,000x4 = 120,000
	0.2	8.94	gas+solid	0.27, 0.31, 0.35, 0.39	15,000x4 = 60,000
			gas+liquid	0.43, 0.47, 0.51, 0.55	15,000x4 = 60,000
			gas	0.59, 0.63, 0.67, 0.71	30,000x4 = 120,000
	0.27	7.70	gas+solid	0.27, 0.31, 0.35, 0.39	15,000x4 = 60,000
			gas+liquid	0.43, 0.47, 0.51, 0.55	15,000x4 = 60,000
			gas	0.59, 0.63, 0.67, 0.71	30,000x4 = 120,000
	0.38	6.49	gas+solid	0.27, 0.31, 0.35, 0.39	15,000x4 = 60,000
			gas+liquid	0.43, 0.47, 0.51, 0.55	15,000x4 = 60,000
			gas	0.59, 0.63, 0.67, 0.71	30,000x4 = 120,000
					Total 1,200,000

Figure 5.2: Initial configurations for data preparations

Figure 5.3 shows the initial position $\mathbf{q}(0)$on gas+solid, gas+liquid, and gas region at each temperature with $\rho = 0.1$ about one sample.

$\mathbf{p}(0)$ are sampled using Boltzmann distribution at fixed temperature in the

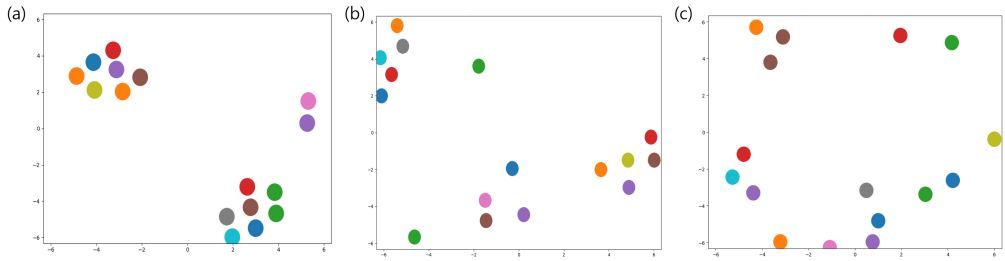

Figure 5.3: Initial position $\mathbf{q}(0)$ on (a) gas+solid, (b) gas+liquid, and (c) gas with $\rho = 0.1$.

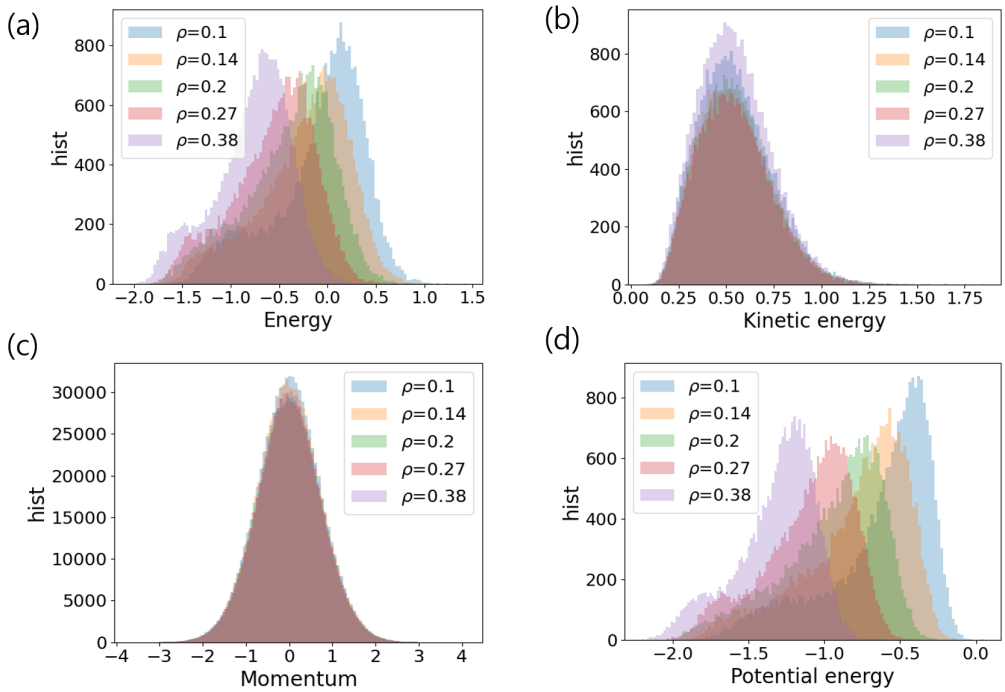

Figure 5.4: Histogram of (a) Energy per particle, (b) Kinetic energy per particle (c) Momentum (d) Potential energy per particle from the initial configurations over all temperatures in $T = [0.59, 0.71]$ with density ρ.

range of $k_BT/J \in [0.27 - 0.71]$. Figure 5.4 shows the histogram of prepared samples as the initial configurations over all temperatures with each density.

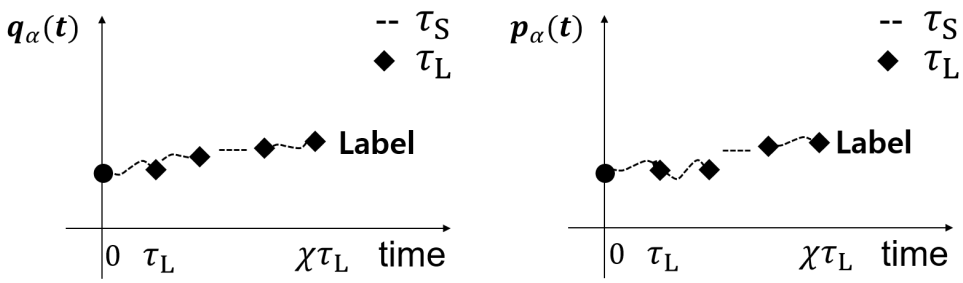

Figure 5.5: A labeled set of training data. τ_S is used to generate $\mathbf{q}_\alpha(\chi m\tau_S)$ and $\mathbf{p}_\alpha(\chi m\tau_S)$ of α-th sample as labels. m is the number of iterations of $\chi\tau_S$ to pair with $\chi\tau_L$.

For each sampled independent configuration, molecular dynamic simulations with velocity-Verlet algorithm in Eq. (3.13) are performed with the time steps $\tau_4 = 10^{-4}$ as small time step. Small time step $\tau_4 = 10^{-4}$ gives high simulation accuracies. Small time step is used to generate accurate data for Deep Learning labels.

5.1.2 Deep Learning to Learn Effective Force

Numerical integrators such as symplectic integrators are effective for small time steps ($\tau \ll 1$, where τ is a dimensionless time unit). However, a large time step causes the numerical integrators to become unstable and particles can overlap, causing artificially large force calculation. To recover the

stability of simulation, we use Deep Learning to learn the effective forces acting on particles. We develop our neural networks to learn effective many-body interactions replacing the force calculations from the original physical model ($-\frac{\partial H}{\partial \mathbf{q}_\alpha}$). Once a new two body and many-body interactions have been learned, a single neural network model can be used to perform accelerated simulations for the Hamiltonian system of interest for arbitrary thermodynamics states (e.g. different temperatures, pressures, number of particles, and densities).

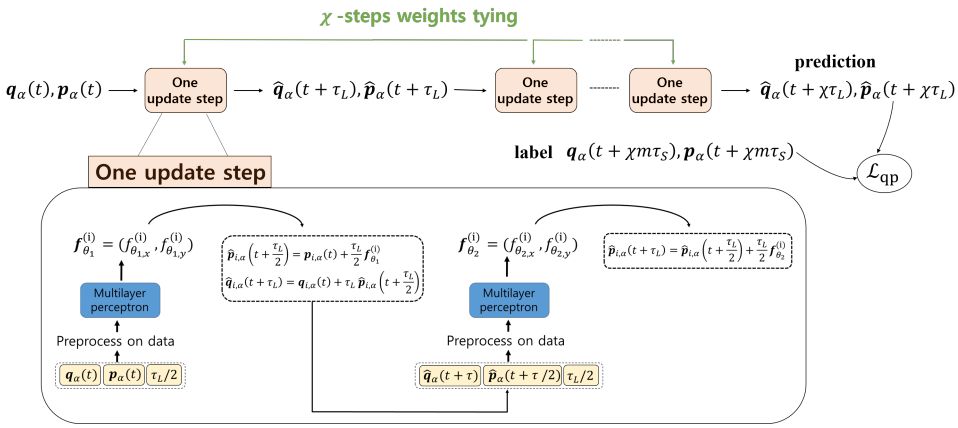

Figure 5.6: Illustration of the χ large time steps neural network forward propagation for symplectic algorithm. Two neural network models for one update step are a MLP and parameterized by θ_1, θ_2 to predict forces $\mathbf{f}_{\theta_1}^{(i)}, \mathbf{f}_{\theta_2}^{(i)}$.

A system of the 2-dimensional Lennard-Jones n particles is considered to produce training data for our fully supervised Deep Learning. Any number of particles can be considered in the experiment. The density of the system is chosen to be various densities ρ and the particles are confined to a

square box of length $L = \sqrt{n/\rho}$. As shown in figure 5.6, our Deep Learning integration algorithm follows a form of the symplectic algorithm, it is parameterized by θ_1, θ_2 to predict effective forces for each i-th particle $\mathbf{f}_{\theta_1}^{(i)}, \mathbf{f}_{\theta_2}^{(i)}$ which is the output of parametric function $\mathbf{f}_{\theta_1}^{(i)} : \Omega \mapsto \mathbb{R}^2$ used in Eq. (5.1) and $\mathbf{f}_{\theta_2}^{(i)} : \Omega \mapsto \mathbb{R}^2$ used in Eq. (5.3), $(\mathbf{q}_\alpha, \mathbf{p}_\alpha) \in \Omega$ is the phase space. Our Deep Learning integrator consists of a series of mathematical manipulation on $(\mathbf{q}_\alpha, \mathbf{p}_\alpha)$ followed by neural network predictions for effective forces for each particle. In this paper, we present several different ways of calculating the effective forces and call these pair-wise neural network (PW-NN), many-body neural network (MB-NN), and combined PW-NN and MB-NN ((PW+MB)-NN). The details will be described in the next subsection. In figure 5.6, we chain up χ large time steps $\chi\tau_L$ to predict the phase space configuration at $(\hat{\mathbf{q}}_\alpha(t+\chi\tau_L), \hat{\mathbf{p}}_\alpha(t+\chi\tau_L))$ and compare to the ground truth labels $(\mathbf{q}_\alpha(t+\chi m\tau_S), \mathbf{p}_\alpha(t+\chi m\tau_S))$. m is the number of iterations of $\chi\tau_S$ to pair with $\chi\tau_L$. The ground truths are generated using the Velocity-Verlet algorithm with very short time step τ_S to get high simulation accuracies whereas our Deep Learning integrators uses large time step τ_L. At each update step, the same parametric functions $\mathbf{f}_{\theta_1}^{(i)}, \mathbf{f}_{\theta_2}^{(i)}$ are used. In another word, the neural network models used to calculate $\mathbf{f}_{\theta_1}^{(i)}, \mathbf{f}_{\theta_2}^{(i)}$ are the same in each one update step. The first neural network model predicts parametric function $\mathbf{f}_{\theta_1}^{(i)}$ to update $\mathbf{p}_{i,\alpha}(t)$ given by $\tau_L/2$ for i-th particle of α-th sample.

$$\hat{\mathbf{p}}_{i,\alpha}(t+\frac{\tau_L}{2}) = \mathbf{p}_{i,\alpha}(t) + \frac{\tau_L}{2}\mathbf{f}_{\theta_1}^{(i)}(\mathbf{q}_\alpha(t), \mathbf{p}_\alpha(t), \frac{\tau_L}{2}) \tag{5.1}$$

After moving $\mathbf{p}_{i,\alpha}(t) \rightarrow \hat{\mathbf{p}}_{i,\alpha}(t+\frac{\tau_L}{2})$, update $\mathbf{q}_{i,\alpha}(t)$ given by τ_L as,

$$\hat{\mathbf{q}}_{i,\alpha}(t+\tau_L) = \mathbf{q}_{i,\alpha}(t) + \tau_L \hat{\mathbf{p}}_{i,\alpha}(t+\frac{\tau_L}{2}) \tag{5.2}$$

The new coordinates $\hat{\mathbf{q}}_\alpha(t+\tau_L), \hat{\mathbf{p}}_\alpha(t+\frac{\tau_L}{2})$ of α-th sample use to prepare the input to the second neural network model which predicts parametric function $\mathbf{f}_{\theta_2}^{(i)}$ to update $\hat{\mathbf{p}}_{i,\alpha}(t+\frac{\tau_L}{2})$ given by $\tau_L/2$.

$$\hat{\mathbf{p}}_{i,\alpha}(t+\tau_L) = \hat{\mathbf{p}}_{i,\alpha}(t+\frac{\tau_L}{2}) + \frac{\tau_L}{2}\mathbf{f}_{\theta_2}^{(i)}(\hat{\mathbf{q}}_\alpha(t+\tau_L), \hat{\mathbf{p}}_\alpha(t+\frac{\tau_L}{2}), \frac{\tau_L}{2}) \tag{5.3}$$

The final configurations $\hat{\mathbf{q}}_\alpha(t+\tau_L), \hat{\mathbf{p}}_\alpha(t+\tau_L)$ at one update step are used for the next update step and update the configurations until χ large time steps.

Computing Parametric Functions $\mathbf{f}_{\theta_1}^{(i)}, \mathbf{f}_{\theta_2}^{(i)}$ for Pairwise Interactions

The force on i-th particle of the LJ system is computed with the pair of its neighboring k-th particle. In Eq. (5.4), the relative position and momentum of neighboring k-th particle with respect to the i-th particle of the α-th sample are considered as input for neural network model for the prediction of pair-wise force between the i-th and k-th particles.

$$\Delta\mathbf{q}_{ik,\alpha} = \mathbf{q}_{i,\alpha} - \mathbf{q}_{k,\alpha}, \quad \Delta\mathbf{p}_{ik,\alpha} = \mathbf{p}_{i,\alpha} - \mathbf{p}_{k,\alpha} \tag{5.4}$$

In figure 5.6, phase space $\mathbf{q}_\alpha$ and $\mathbf{p}_\alpha$ of α sample at time t prepare the input to feed into the first neural network model as $\Delta\mathbf{q}_{ik,\alpha}$, $\Delta\mathbf{p}_{ik,\alpha}$, and large time step $\frac{\tau_L}{2}$. The function $\mathbf{f}_{\theta_1}^{(i)}(\mathbf{q}_\alpha(t), \mathbf{p}_\alpha(t), \frac{\tau_L}{2})$ is computed as the sum of the

predicted individual force on i-th particle with a pair of its neighboring k-th particle as shown in Eq. (5.5).

$$\mathbf{f}_{\theta_1}^{(i)} = \sum_{k \neq i} \mathbf{pw\text{-}net}_{\theta_1}(\Delta\ \mathbf{q}_{ik,\alpha}(t), \Delta\mathbf{p}_{ik,\alpha}(t), \frac{\tau_L}{2}) \tag{5.5}$$

The new configurations $\hat{\mathbf{q}}_\alpha(t+\tau_L), \hat{\mathbf{p}}_\alpha(t+\frac{\tau_L}{2})$ are calculated from $\mathbf{f}_{\theta_1}^{(i)}$ using Eq. (5.1) and (5.2), and use to prepare the input to the second neural network model as $\Delta\hat{\mathbf{q}}_{ik,\alpha}(t+\tau_L), \Delta\hat{\mathbf{p}}_{ik,\alpha}(t+\frac{\tau_L}{2})$, and large time step $\frac{\tau_L}{2}$. In the same way, the function $\mathbf{f}_{\theta_2}^{(i)}(\hat{\mathbf{q}}_\alpha(t+\tau_L), \hat{\mathbf{p}}_\alpha(t+\frac{\tau_L}{2}), \frac{\tau_L}{2})$ is computed from the individual force output in Eq. (5.6).

$$\mathbf{f}_{\theta_2}^{(i)} = \sum_{k \neq i} \mathbf{pw\text{-}net}_{\theta_2}(\Delta\hat{\mathbf{q}}_{ik,\alpha}(t+\tau_L), \Delta\hat{\mathbf{p}}_{ik,\alpha}(t+\frac{\tau_L}{2}), \frac{\tau_L}{2}) \tag{5.6}$$

The final configurations $\hat{\mathbf{q}}_\alpha(t+\tau_L), \hat{\mathbf{p}}_\alpha(t+\tau_L)$ using Eq. (5.3) complete one integration step.

Computing Parametric Functions $\mathbf{f}_{\theta_1}^{(i)}, \mathbf{f}_{\theta_2}^{(i)}$ for Many-Body Interactions

To calculate many-body interactions of a i-th particle with a set of neighboring particles $\{k_j\}$, a hexagonal grid (see Figure 5.7) centered at the i-th particle is created. Let $\boldsymbol{u}_{i,l,\alpha}$ be the l-th grid position of i-th particle of α-th sample, $l = 1, ..., 6$. $\boldsymbol{u}_{i,l,\alpha}$ has two components $(u_{i,l,x,\alpha}, u_{i,l,y,\alpha})$. Given a vector of position $\boldsymbol{u}_{i,l,\alpha}$ of l-th grid point of the α-th sample confined to a square box, a derivative potential $\nabla u_{i,l,\alpha}\Phi = (\frac{\partial\Phi}{\partial u_{i,l,x,\alpha}}, \frac{\partial\Phi}{\partial u_{i,l,y,\alpha}})$ gives the force field on the l-th grid position of i-th particle, interacting via potential

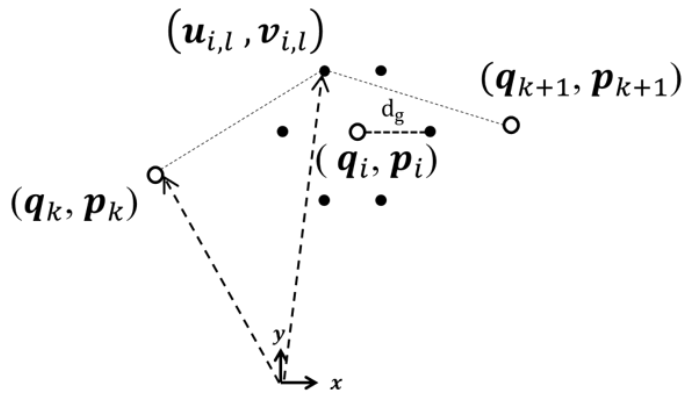

Figure 5.7: l-th grid position $\boldsymbol{u}_{i,l}$ centered at the i-th particle created to calculate many-body interactions of a i-th particle.

ϕ^{LJ} defined in Eq. (3.6).

$$\nabla u_{i,l,\alpha}\Phi(\boldsymbol{u}_{i,l,\alpha}, \mathbf{q}_{1,\alpha}, ..., \mathbf{q}_{n,\alpha}) = \sum_{k\neq i} \nabla\phi^{LJ}(|\boldsymbol{u}_{i,l,\alpha} - \mathbf{q}_{k,\alpha}|) \tag{5.7}$$

If the distance of $|\boldsymbol{u}_{i,l,\alpha} - \mathbf{q}_{k,\alpha}|$ is too close, derivative potential energy calculates large value and makes training for Deep Learning hard. Each component of $\nabla u_{i,l,\alpha}\Phi$ is clipped to a threshold ω in Eq.(5.8) to prevent the artificially large value of derivative potential energy . We set $\omega = 108.35$ which is derivative potential energy with $|\boldsymbol{u}_{i,l,\alpha} - \mathbf{q}_{k,\alpha}| = 0.9$.

$$\left|\frac{\partial\Phi}{\partial u_{i,l,x,\alpha}}\right| = \left|\frac{\partial\Phi}{\partial u_{i,l,y,\alpha}}\right| = \omega \tag{5.8}$$

The momentum field $\boldsymbol{v}_{i,l,\alpha}$ of the grid points of α-th sample can be obtained to average of weights $w^k_{i,l,\alpha}$ factored by the velocity of k-th particle

$$\boldsymbol{v}_{i,l,\alpha} = \frac{\sum_{k=1} w^k_{i,l,\alpha}\mathbf{p}_{k,\alpha}}{\sum_{k=1} w^k_{i,l,\alpha}} \tag{5.9}$$

where $\mathbf{p}_{k,\alpha}$ is monentum of k-th particle of α-th sample. The weight $w^k_{i,l,\alpha}$ is defined as the inverse proportion of the distance between the $\boldsymbol{u}_{i,l,\alpha}$ and $\mathbf{q}_{k,\alpha}$ in Eq. (5.10).

$$w^k_{i,l,\alpha} = \frac{1}{|\boldsymbol{u}_{i,l,\alpha} - \mathbf{q}_{k,\alpha}|} \tag{5.10}$$

In figure 5.6, data is $\mathbf{q}_\alpha$, $\mathbf{p}_\alpha$ of α-th sample at time t and input into first neural network model is considered as $\{\nabla u_{i,l,\alpha}\Phi\}$, $\{\boldsymbol{v}_{i,l,\alpha} - \mathbf{p}_{i,\alpha}\}$, and large time step $\frac{\tau_L}{2}$. The relative momentum $\boldsymbol{v}_{i,l,\alpha} - \mathbf{p}_{i,\alpha}$ causes the information of momentum and preserve the difference from the center of i-th particle as the reference point regardless of the independent configurations such as the translation invariance. The output is the function $\mathbf{f}^{(i)}_{\theta_1}(\mathbf{q}_\alpha(t), \mathbf{p}_\alpha(t), \frac{\tau_L}{2})$ computed for i-th particle of α-th sample as shown in Eq. (5.11).

$$\mathbf{f}^{(i)}_{\theta_1} = \textbf{mb-net}_{\theta_1}(\{\nabla u_{i,l,\alpha}\Phi(t)\}, \{\boldsymbol{v}_{i,l,\alpha}(t) - \mathbf{p}_{i,\alpha}(t)\}, \frac{\tau_L}{2}) \tag{5.11}$$

The second neural network model uses the input as $\{\nabla u_{i,l,\alpha}\hat{\Phi}(t+\tau_L)\}$, $\{\hat{\boldsymbol{v}}_{i,l,\alpha}(t+\frac{\tau_L}{2}) - \hat{\mathbf{p}}_{i,\alpha}(t+\frac{\tau_L}{2})\}$, and large time step $\frac{\tau_L}{2}$ with the new coordinates $\hat{\mathbf{q}}_{i,\alpha}(t+\tau_L), \hat{\mathbf{p}}_{i,\alpha}(t+\frac{\tau_L}{2})$ calculated from the output using Eq. (5.1) and Eq. (5.2).

$$\mathbf{f}^{(i)}_{\theta_2} = \textbf{mb-net}_{\theta_2}(\{\nabla u_{i,l,\alpha}\hat{\Phi}(t+\tau_L)\}, \{\hat{\boldsymbol{v}}_{i,l,\alpha}(t+\frac{\tau_L}{2}) - \hat{\mathbf{p}}_{i,\alpha}(t+\frac{\tau_L}{2})\}, \frac{\tau_L}{2}) \tag{5.12}$$

The final configurations $\hat{\mathbf{q}}_\alpha(t+\tau_L), \hat{\mathbf{p}}_\alpha(t+\tau_L)$ by the output $\mathbf{f}^{(i)}_{\theta_2}(\hat{\mathbf{q}}_\alpha(t+\tau_L), \hat{\mathbf{p}}_\alpha(t+\frac{\tau_L}{2}), \frac{\tau_L}{2})$ in Eq. (5.12) using Eq. (5.3) complete one integration step.

Combined PW-NN and MB-NN : (PW+MB)-NN

(PW+MB)-NN is combined with PW-NN and MB-NN. Each neural network model to predict effective force, $\mathbf{f}_{\theta_1}^{(i)}(\mathbf{q}_\alpha(t), \mathbf{p}_\alpha(t), \frac{\tau_L}{2})$ and $\mathbf{f}_{\theta_2}^{(i)}(\hat{\mathbf{q}}_\alpha(t+\tau_L), \hat{\mathbf{p}}_\alpha(t+\frac{\tau_L}{2}), \frac{\tau_L}{2})$, has the parametric function as follows:.

$$\begin{aligned}\mathbf{f}_{\theta_1}^{(i)} &= \Big(\Gamma_1 \mathbf{mb\text{-}net}_{\theta_1}(\{\nabla u_{i,l,\alpha}\Phi(t)\}, \{\boldsymbol{v}_{i,l,\alpha}(t) - \mathbf{p}_{i,\alpha}(t)\}, \frac{\tau_L}{2}) \\ &+ (1-\Gamma_1)\sum_k \mathbf{pw\text{-}net}_{\theta_1}(\Delta\mathbf{q}_{ik,\alpha}(t), \Delta\mathbf{p}_{ik,\alpha}(t), \frac{\tau_L}{2})\Big) \qquad (5.13)\end{aligned}$$

$$\begin{aligned}\mathbf{f}_{\theta_2}^{(i)} &= \Big(\Gamma_2 \mathbf{mb\text{-}net}_{\theta_2}(\{\nabla u_{i,l,\alpha}\hat{\Phi}(t+\tau_L)\}, \{\hat{\boldsymbol{v}}_{i,l,\alpha}(t+\frac{\tau_L}{2}) - \hat{\mathbf{p}}_{i,\alpha}(t+\frac{\tau_L}{2})\}, \frac{\tau_L}{2}) \\ &+ (1-\Gamma_2)\sum_k \mathbf{pw\text{-}net}_{\theta_2}(\Delta\hat{\mathbf{q}}_{ik,\alpha}(t+\tau_L), \Delta\hat{\mathbf{p}}_{ik,\alpha}(t+\frac{\tau_L}{2}), \frac{\tau_L}{2})\Big) \qquad (5.14)\end{aligned}$$

where Γ_1 and Γ_2 are the learning parameters with sigmoid function and represent the weights of MB-NN to learn the effective forces compared with PW-NN.

5.1.3 Loss Functions

The loss function is defined in Eq. (5.15) as follows:

$$L(\theta_1, \theta_2, \theta_3) = L_{qp}(\theta_1, \theta_2) + w_e L_e(\theta_3) \qquad (5.15)$$

$$L_{qp}(\theta_1, \theta_2) = L_q(\theta_1, \theta_2) + L_p(\theta_1, \theta_2)$$

$$L_q(\theta_1, \theta_2) = \frac{1}{N}\sum_\alpha^N \|\hat{\mathbf{q}}_\alpha(\chi\tau_L) - \mathbf{q}_\alpha(\chi m\tau_S)\|_2$$

$$L_p(\theta_1, \theta_2) = \frac{1}{N}\sum_\alpha^N \|\hat{\mathbf{p}}_\alpha(\chi\tau_L) - \mathbf{p}_\alpha(\chi m\tau_S)\|_2$$

$$L_e(\theta_3) = \left|\hat{E}(\chi\tau_L) - E(0)\right|$$

here, N is the training data set. The first term of loss function for one data point of $(\mathbf{q}_\alpha(0), \mathbf{p}_\alpha(0), \mathbf{q}_\alpha(\chi m\tau_S), \mathbf{p}_\alpha(\chi m\tau_S))$ is the mean-square error. L_{qp} minimizes difference between predictions at χ large time steps and labels at a number of short time steps that pair with χ large time steps. By adding a second term L_e, conservation of energy is learned from initial energy and becomes small when $\hat{E}(\chi\tau_L)$ is close to the $E(0)$ during training. w_e is the weight of the second term of the loss.

5.1.4 Sequence of Molecular Dynamics with Deep Learning

To deploy the trained PW-NN, MB-NN, and (PW+MB)-NN for molecular dynamics, a sequence of molecular dynamics are conducted every χ large time steps at a given large time step $\tau_L = 0.1$. The effective forces are predicted by deep learning model and predict the new configurations ($\hat{\mathbf{q}}_\alpha(\chi\tau_L)$, $\hat{\mathbf{p}}_\alpha(\chi\tau_L)$) by using the predicted forces. With the new configurations, the configurations ($\hat{\mathbf{q}}_\alpha(2\chi\tau_L), \hat{\mathbf{p}}_\alpha(2\chi\tau_L)$) at next χ-time steps are predicted. The new time integration scheme is repeated up to maximum time t_{max}. This scheme requires only forward propagation using the GPU implemented with PyTorch library [46], so the computing time is negligible.

5.2 Results and Discussions

Our proposed PW-NN, MB-NN, and (PW+MB)-NN is used to recover the stability of the simulation and speed up time integration using large time

step by predicting the effective forces. The effectiveness of our PW-NN, MB-NN, and (PW+MB)-NN is evaluated on gas+solid regions at $k_BT/J = 0.27$, gas+liquid regions at $k_BT/J = 0.47$, and gas regions at $k_BT/J = 0.71$ with various densities $\rho = 0.1, 0.14, 0.2, 0.27$, and 0.38 in the phase diagram of Lennard-Jones systems. $\gamma = 1,000$ independent configurations are generated for use as test data sets at each temperature. The velosity-verlet algorithm is used for all experiments.

5.2.1 Accumulate Function of Time

When two particles interacting with the LJ potential are nearly overlapping, the forces can compute the artificially large value. To check the stability of the simulation at large time step, we find the threshold distance q_{thrsh} using the minimum distance about all samples and set the threshold force f_{thrsh} by instituting q_{thrsh} for the force on LJ potential formula in Eq. (5.16).

$$f_{thrsh} = -4\left((-12)\frac{1}{q_{thrsh}^{13}} - (-6)\frac{1}{q_{thrsh}^{7}}\right) \tag{5.16}$$

A number of unstable configurations that have more than threshold force f_{thrsh} over time are counted. We set the $q_{thrsh} = 0.7259$, causing the large force calculation. The force $\mathbf{f}_k$ on the k-th particle for all pairs has components f_{kx}, f_{ky}. The magnitude of force per particle is calculated as $|\ \mathbf{f}_k'| = \frac{|\ \mathbf{f}_k|}{n}$. k=1,...,n, n is the number of particles. The number of the unstable configurations that have the maximum value $f_0^{'} = \max(|\mathbf{f}_k'|)$ of n particles more than f_{thrsh} are accumulated over time t for a maximum time of $t_{max} = 1000$

at $k_BT/J = 0.27, 0.47,$ and 0.71 as shown in figure 5.8.

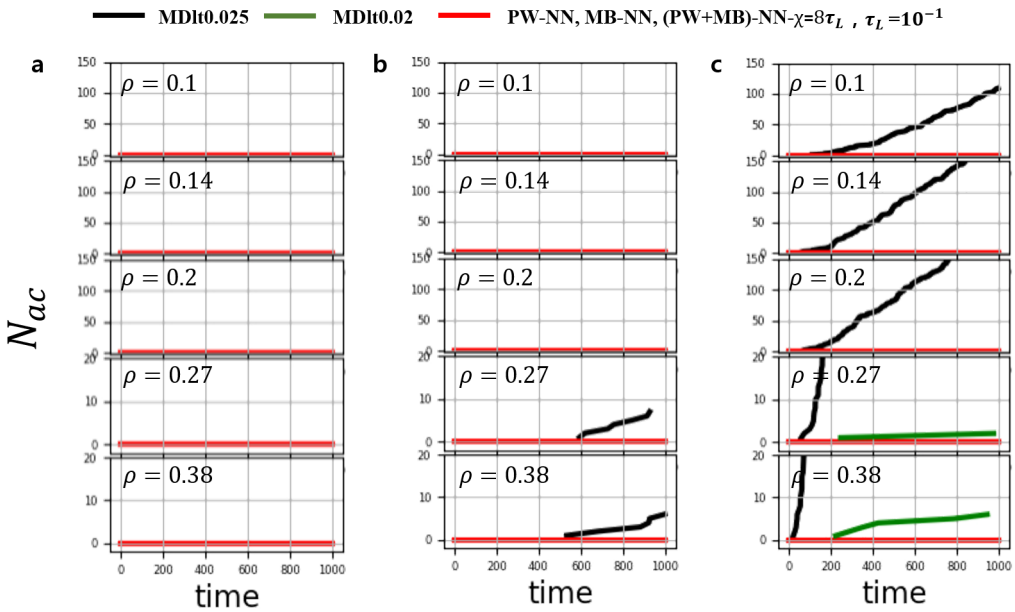

Figure 5.8: Accumulate function of time for a maximum time of $t_{max} = 1000$ at **a**, $k_BT/J = 0.27$ on gas+solid regions, **b**, $k_BT/J = 0.47$, on gas+liquid regions, and **c**, $k_BT/J = 0.71$ on gas regions with various densities ρ.

The maximum of a number of configurations from instability of simulation is 1000 as the test data sets. Overall, the MD simulations at $\tau = 0.025$ (black) and $\tau = 0.02$ (green) show that a number of the unstable configurations are accumulated with a various density near critical temperature and at high temperature. On the gas region at high temperature, the particles move fast and easily come much closer together. At time step tau 0.025, the unstable configurations are accumulated early with a various density and also accumulated at tau=0.02. The Deep Learning results at $\tau = 0.1$ for PW-NN,

MB-NN, and (PW+MB)-NN show the stability of simulation. The configurations is not accumulated over time at each temperature with various densities.

5.2.2 Distance Metric

To evaluate the accuracy of the simulation results, the mean square error between two trajectories is investigated by comparing with highly accurate molecular dynamics trajectory $(\mathbf{q}_\alpha^{\tau'}(t), \mathbf{p}_\alpha^{\tau'}(t))$ performed with $\tau' = 10^{-4}$. $\tau' = 10^{-4}$ is used as the reference time step. Let $\mathbf{q}_{i,\alpha}^{\tau}(t) = (q_{i,x,\alpha}^{\tau}(t), q_{i,y,\alpha}^{\tau}(t))$ and $\mathbf{p}_{i,\alpha}^{\tau}(t) = (p_{i,x,\alpha}^{\tau}(t), p_{i,y,\alpha}^{\tau}(t))$ be the α^{th} trajectory of i-th particle integrated over time step τ. $i = 1 \cdots n$, n is the number of particles. For comparing two integration schemes with time steps τ and τ', the mean square error between two trajectories is defined as follows.

$$\Delta_\alpha^{\tau,\tau'}(t) = \frac{1}{n}\sum_i^n \left(\mathbf{q}_{i,\alpha}^{\tau}(t) - \mathbf{q}_{i,\alpha}^{\tau'}(t)\right)^2 + \frac{1}{n}\sum_i^n \left(\mathbf{p}_{i,\alpha}^{\tau}(t) - \mathbf{p}_{i,\alpha}^{\tau'}(t)\right)^2 \quad (5.17)$$

We estimate the mean of the $\gamma = 1000$ mean square error between two trajectories as function of time at each temperature.

Figure 5.9 shows that the mean of mean square error between two trajectories $\bar{\Delta}^{\tau,\tau'}(t)$ over time t for a maximum time of $t_{max} = 1000$ at $k_BT/J =$ 0.27, 0.47, and 0.71. Each figure shows the mean square error results at different temperature with various densities. The MD simulation at time step $\tau = 0.01$ (black) shows that the mean square error at low temperature is lower than higher temperature with various densities. The results of PW-NN

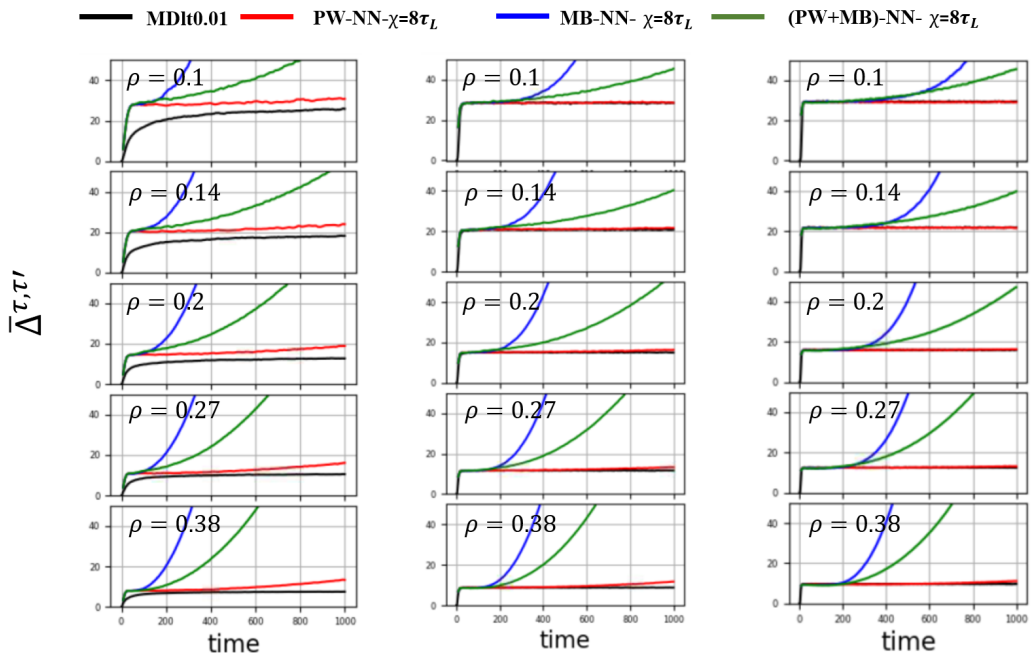

Figure 5.9: The mean of correlation values $\bar{\Delta}^{\tau,\tau'}(t)$ for a maximum time of $t_{max} = 1000$ at $\mathbf{a}, k_BT/J = 0.27$, $\mathbf{b}, k_BT/J = 0.47$, and $\mathbf{c}, k_BT/J = 0.71$ with various densities.

(red) show that the mean square error is similar results with MD results at time step $\tau = 0.01$ at each temperature with various densities than other NNs methods. the PW-NN with time step $\tau = 0.1$ are similar accuracy as the MD simulation with time step $\tau = 0.01$.

5.2.3 Conservation of Energy

We investigate the conservation of energy by calculating the total energy as function of time. To calculate the mean of energy over time about the all samples, energy of the initial configurations are shifted to zero. because the

starting configurations across samples are different. Eq. (5.18) show how we shift the energy $e_\alpha(t)$ per α sample at each time step t. Here, γ represents the number of samples at each temperature.

$$\tilde{e}_\alpha(t) = e_\alpha(t) - e_\alpha(0) \qquad \alpha = 1, \ldots, \gamma. \tag{5.18}$$

With the shifting of energy, we compute the mean of energy per particle $\mu_{\tilde{e}}(t)$ over independent samples.

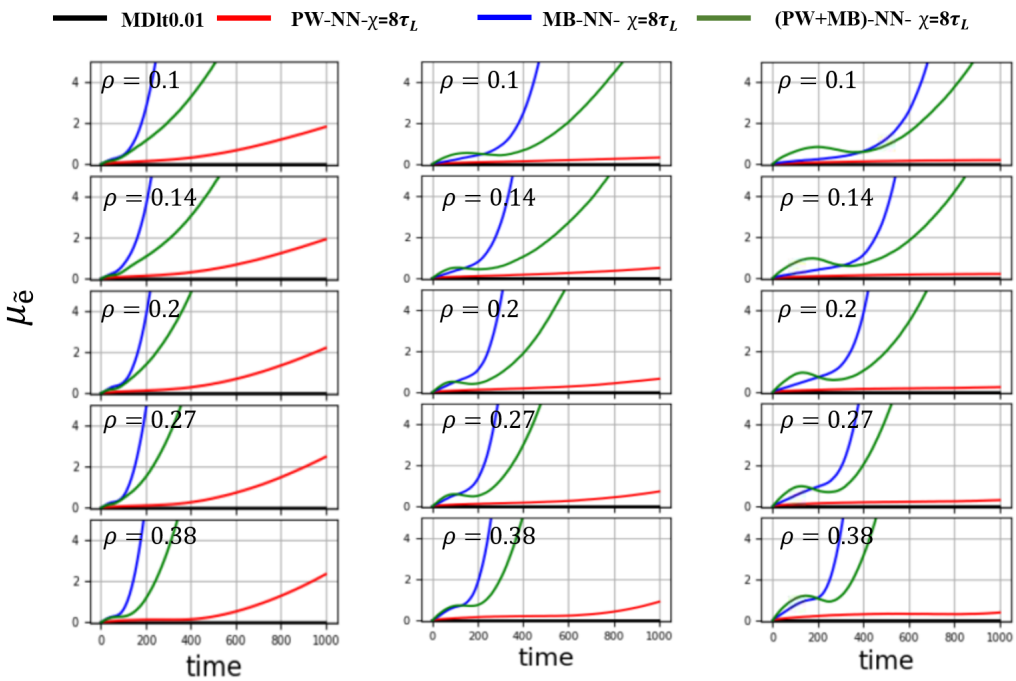

Figure 5.10: Energy conservations for a maximum time of $t_{max} = 1000$ at **a**, $k_BT/J = 0.27$, **b**, $k_BT/J = 0.47$, and **c**, $k_BT/J = 0.71$ with various densities.

Figure 5.10 shows the mean of energy results at different temperature with various densities. Overall, for MD simulation with time steps $\tau = 0.01$,

conservation of energy is good compared with the Deep Learning results. The results of PW-NN show that the mean of energy is lower than other NN methods over time. Our Deep Learning models are still training and have not reached convergence yet so if they train more, the results will show that the conservation of energy is recovered.

Chapter 6

Conclusion

Our results have demonstrated that the deep learning corrections enhance the time integration step of the symplectic method. In spin dynamic simulations, our DL method has achieved ~ 10 times computational speed up while maintaining accuracy compared to the original Suzuki-Trotter decomposition method. The natural of local nearest neighbours interactions in the lattice means that convolutional structure of the Deep Neural Network is a nature choice of network architecture. Since convolution is translationally invariant, the effect of lattice size on training our U-Net is not a major concern. For example, between $L = 4$ and $L = 8$ lattices, the time required for training the U-Net parameters increases by about 4 times, which is sublinear with respect to the number of lattice sites. Our Deep Learning was trained on simulation data at $\tau = 10^{-3}$, however, its accuracy performance is equivalent to simulation data at $\tau = 10^{-2}$. This shows that our Deep

CHAPTER 6. CONCLUSION

Learning training has not reached its theoretical limit of a perfect prediction. This theoretical limit can be achieved exactly if we train on an infinite amount of data for an infinite capacity. In practice, Deep Learning methods can not be perfect because the amount of data and the capacity of U-Net are finite. The main source of inaccuracies in our Deep Learning method is that U-Net's output does not fit exactly the labeled data generated at $\tau = 10^{-3}$ and that even if U-Net is able to fit the data it has through training, it may not predict perfectly on the data it has never seen in training. In the study of molecular dynamic simulations, Our results using PW-NN have demonstrated that the effective force obtained by using Deep Learning with large time step $\tau = 0.1$ recovers the stability of simulation and produces similar accuracy as the MD simulation with integration time step $\tau = 0.01$. With other NN methods, MB-NN and (PW+MB)-NN, training more longer to reach convergence, our Deep Learning simulations will show that about 10x speed up of the simulation. We can apply the Deep Learning driven MD simulations to the study of protein simulations (protein-protein interactions, protein-ligand interactions, etc) that it takes long time.

References

[1] D. P. Landau and M. Krech, "Spin dynamics simulations of classical ferro- and antiferromagnetic model systems: comparison with theory and experiment," *Journal of Physics: Condensed Matter*, vol. 11, pp. R179–R213, jan 1999.

[2] J. W. Lynn, "Temperature dependence of the magnetic excitations in iron," *Phys. Rev. B*, vol. 11, pp. 2624–2637, Apr 1975.

[3] O. Gutfleisch, M. A. Willard, E. Brück, C. H. Chen, S. G. Sankar, and J. P. Liu, "Magnetic Materials and Devices for the 21st Century: Stronger, Lighter, and More Energy Efficient," *Advanced Materials*, vol. 23, pp. 821–842, Feb. 2011.

[4] S. Sugimoto, "Current status and recent topics of rare-earth permanent magnets," *Journal of Physics D: Applied Physics*, vol. 44, p. 064001, Feb. 2011.

[5] J. Slaughter, "Materials for Magnetoresistive Random Access Memory," *Annual Review of Materials Research*, vol. 39, pp. 277–296, Aug. 2009.

[6] J.-Y. Bigot and M. Vomir, "Ultrafast magnetization dynamics of nanostructures: Ultrafast magnetization dynamics of nanostructures," *Annalen der Physik*, vol. 525, pp. 2–30, Feb. 2013.

[7] J. Walowski and M. Münzenberg, "Perspective: Ultrafast magnetism and THz spintronics," *Journal of Applied Physics*, vol. 120, p. 140901, Oct. 2016.

[8] H. K. Lee and Z. Yuan, "Studies of the magnetization reversal process driven by an oscillating field," *Journal of Applied Physics*, vol. 101, no. 3, p. 033903, 2007.

[9] M. Kryder, E. Gage, T. McDaniel, W. Challener, R. Rottmayer, Ganping Ju, Yiao-Tee Hsia, and M. Erden, "Heat Assisted Magnetic Recording," *Proceedings of the IEEE*, vol. 96, pp. 1810–1835, Nov. 2008.

[10] H. K. Lee and Y. Okabe, "Exchange bias with interacting random antiferromagnetic grains," *Physical Review B*, vol. 73, p. 140403, Apr. 2006.

[11] B. J. Alder and T. E. Wainwright, "Studies in molecular dynamics. i. general method," *The Journal of Chemical Physics*, vol. 31, no. 2, pp. 459–466, 1959.

[12] R. J. Bearman and P. F. Jones, "Statistical mechanical theory of the viscosity coefficients of binary liquid solutions," *The Journal of Chemical Physics*, vol. 33, no. 5, pp. 1432–1438, 1960.

[13] G. Bussi and M. Parrinello, "Accurate sampling using langevin dynamics," *Phys. Rev. E*, vol. 75, p. 056707, May 2007.

[14] S. Ranganathan, G. S. Dubey, and K. N. Pathak, "Molecular-dynamics study of two-dimensional lennard-jones fluids," *Phys. Rev. A*, vol. 45, pp. 5793–5797, Apr 1992.

[15] H. Zhao and A. Caflisch, "Molecular dynamics in drug design," *Eur. J. Med. Chem.*, vol. 91, pp. 4–14, Feb. 2015.

[16] K. Lindorff-Larsen, S. Piana, R. O. Dror, and D. E. Shaw, "How fast-folding proteins fold," *Science*, vol. 334, no. 6055, pp. 517–520, 2011.

[17] D. Frenkel and B. Smit, *Understanding molecular simulation : from algorithms to applications. 2nd ed*, vol. 50. 01 1996.

[18] R. D. S. Benedict J. Leimkuhler, Sebastian Reich, "Some multistep methods for use in molecular dynamics calculations," *Journal of Computational Physics*, vol. 20, no. 2, pp. 130 – 139, 1976.

[19] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*. USA: Clarendon Press, 1988.

[20] S. Kim, "Time step and shadow Hamiltonian in molecular dynamics simulations," *Journal of the Korean Physical Society*, vol. 67, pp. 418–422, Aug. 2015.

[21] R. D. Engle, R. D. Skeel, and M. Drees, "Monitoring energy drift with shadow Hamiltonians," *Journal of Computational Physics*, vol. 206, pp. 432–452, July 2005.

[22] J. Carrasquilla and R. G. Melko, "Machine learning phases of matter," *Nature Physics*, vol. 13, pp. 431–434, May 2017.

[23] W. Zhang, J. Liu, and T.-C. Wei, "Machine learning of phase transitions in the percolation and X Y models," *Physical Review E*, vol. 99, p. 032142, Mar. 2019.

[24] Z. Li, M. Luo, and X. Wan, "Extracting critical exponents by finite-size scaling with convolutional neural networks," *Physical Review B*, vol. 99, p. 075418, Feb. 2019.

[25] E. van Nieuwenburg, Y.-H. Liu, and S. Huber, "Learning phase transitions by confusion," *Nature Physics*, vol. 13, pp. 435–439, May 2017.

[26] A. Morningstar and R. G. Melko, "Deep learning the ising model near criticality," *J. Mach. Learn. Res.*, vol. 18, p. 5975–5991, Jan. 2017.

[27] J. Greitemann, K. Liu, and L. Pollet, "Probing hidden spin order with interpretable machine learning," *Phys. Rev. B*, vol. 99, p. 060404, Feb 2019.

[28] L. Huang and L. Wang, "Accelerated Monte Carlo simulations with restricted Boltzmann machines," *Physical Review B*, vol. 95, p. 035105, Jan. 2017.

[29] D. C. Rapaport, *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 2004.

[30] F. Noé, S. Olsson, J. Köhler, and H. Wu, "Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning," *Science*, vol. 365, p. eaaw1147, Sept. 2019.

[31] G. Fabiani and J. H. Mentink, "Investigating ultrafast quantum magnetism with machine learning," *SciPost Phys.*, vol. 7, p. 4, 2019.

[32] P. Weinberg and M. Bukov, "QuSpin: a Python Package for Dynamics and Exact Diagonalisation of Quantum Many Body Systems part I: spin chains," *SciPost Phys.*, vol. 2, p. 003, 2017.

[33] Y. A. Kharkov, V. E. Sotskov, A. A. Karazeev, E. O. Kiktenko, and A. K. Fedorov, "Revealing quantum chaos with machine learning," *Phys. Rev. B*, vol. 101, p. 064406, Feb 2020.

[34] G. Carleo and M. Troyer, "Solving the quantum many-body problem with artificial neural networks," *Science*, vol. 355, no. 6325, pp. 602–606, 2017.

[35] S. Park, W. Kwak, and H. K. Lee, "Accelerated spin dynamics using deep learning corrections," *Scientific Reports*, vol. 10, p. 13772, Aug 2020.

[36] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, vol. 21, pp. 1087–1092, June 1953.

[37] K. Chen, A. M. Ferrenberg, and D. P. Landau, "Static critical behavior of three-dimensional classical Heisenberg models: A high-resolution Monte Carlo study," *Physical Review B*, vol. 48, pp. 3249–3256, Aug. 1993.

[38] M. R. Reddy and S. F. O'Shea, "The equation of state of the two-dimensional lennard–jones fluid," *Canadian Journal of Physics*, vol. 64, no. 6, pp. 677–684, 1986.

[39] R. R. Singh, K. S. Pitzer, J. J. de Pablo, and J. M. Prausnitz, "Monte carlo simulation of phase equilibria for the two-dimensional lennard-jones fluid in the gibbs ensemble," *The Journal of Chemical Physics*, vol. 92, no. 9, pp. 5463–5466, 1990.

[40] S. Ranganathan and K. N. Pathak, "Freezing transition of two-dimensional lennard-jones fluids," *Phys. Rev. A*, vol. 45, pp. 5789–5792, Apr 1992.

[41] S.-H. Tsai, H. K. Lee, and D. P. Landau, "Molecular and spin dynamics simulations using modern integration methods," *American Journal of Physics*, vol. 73, pp. 615–624, July 2005.

[42] K. Binder, "The Monte Carlo method for the study of phase transitions: A review of some recent progress," *Journal of Computational Physics*, vol. 59, pp. 1–55, May 1985.

[43] T. Paauw, A. Compagner, and D. Bedeaux, "Monte-Carlo calculation for the classical F.C.C. Heisenberg ferromagnet," *Physica A: Statistical Mechanics and its Applications*, vol. 79, pp. 1–17, Jan. 1975.

[44] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *arXiv:1505.04597*, May 2015.

[45] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals,

P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv:1603.04467*, 2016.

[46] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.