2022-12-01

Generation Of Phase Transitions Boundaries Via Convolutional Neural Networks

Christopher Alexis Ibarra
University of Texas at El Paso

GENERATION OF PHASE TRANSITIONS BOUNDARIES VIA CONVOLUTIONAL NEURAL NETWORKS

CHRISTOPHER ALEXIS IBARRA

Master's Program in Computational Science

APPROVED:

Ramon Ravelo, Ph.D., Chair

Jorge A. Munoz, Ph.D.

Vladik Kreinovich, Ph.D.

Stephen Crites, Ph.D.
Dean of the Graduate School

GENERATION OF PHASE TRANSITIONS BOUNDARIES VIA CONVOLUTIONAL NEURAL NETWORKS

by

CHRISTOPHER ALEXIS IBARRA

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Computational Science Program

THE UNIVERSITY OF TEXAS AT EL PASO

December 2022

Acknowledgements

I would like to thank my advisor Dr. Ramon Ravelo for giving me the project this work was based on. I would also like to thank all my instructors from the PhD Computational Science program at the University of Texas at El Paso for showing me the algorithms, numerical methods, and programming skills I applied in this work.

Abstract

Accurate mapping of phase transitions boundaries is crucial in accurately modeling the equation of state of materials. The phase transitions can be structural (solid-solid) driven by temperature or pressure or a phase change like melting which defines the solid-liquid melt line. There exist many computational methods for evaluating the phase diagram at a particular point in temperature (T) and pressure (P). Most of these methods involve evaluation of a single (P,T) point at a time. The present work partially automates the search for phase boundaries lines utilizing a machine learning method based on convolutional neural networks and an efficient search algorithm and a shrinking enclosure. This neural network (NN) approach is applied to the prediction of the melt line of metals as a function of pressure. The proposed NN method is implemented using the so-called Z-method [1], a molecular-dynamics-based computational approach for determining upper bounds in the solid-liquid melt line of a material. In this method, the system is subjected to "jumps" in temperature until melting is achieved. The usefulness of our proposed NN search method is that it can be easily applied to a wide range of inter-atomic potentials and hence help test their accuracy and agreement with experiments. Future machine learning applications can be similarly applied for determining more subtle and complex phase diagrams.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In many applications, it is important to know how the melting temperature of various metals depends on the pressure. This dependence is known as the *melting line*. In principle, it is possible to measure melting temperature for all the pressure values, but for high pressures, such a measurement is very difficult and expensive to perform. So, to find the melting line, researchers come up with numerical models describing melting, and use these models to determine the melting line.

In this thesis, we propose a more efficient method of determining the melting line of various metals (such as copper0 that uses convolutional neural networks, the binary search algorithm, and a shrinking enclosure. These machine learning, search algorithm, and interval computation methods or similar can be used to produce more detailed phase transition lines between solid phases of crystal metals and higher dimensional phase diagrams.

In this work, by neural networks, we will mean artificial neural networks as opposed to biological neural networks. Neural networks are a machine learning method for interpolation. A neural network aims to guess the correct output to an input. A particular kind of neural network is the so-called convolutional neural network, which has a topology suited for image recognition, where an image is the input and a categorical value, or perhaps a corresponding integer, is the output.

In this work we will focus on the so-called z-method [1]. In this method, a system with periodic boundary conditions is set to a super-heated critical temperature T_{LS} until the system melts and the temperature drops to the melting temperature T_m.

Although the z-method is considered for the application convolutional neural networks, there are other methods for finding the melting line of metals. The z-method goes from solid

to liquid across the melting line, but going from liquid to solid (solidification or freezing) can also be used to map the melting line. The process involves starting at a liquid state and applying pressure until solidification occurs. A work on this [2] applied pressure on liquid tantalum for spontaneous crystal nucleation as shown in Figure 1.1. This method was applied on systems of 64K and 16M tantalum atoms, with periodic boundary conditions, and replicated the smaller system, contiguous to each other, for same size comparison. The replicated smaller system crystals did not coincide with that of the larger system crystals as shown in Figure 1.2.

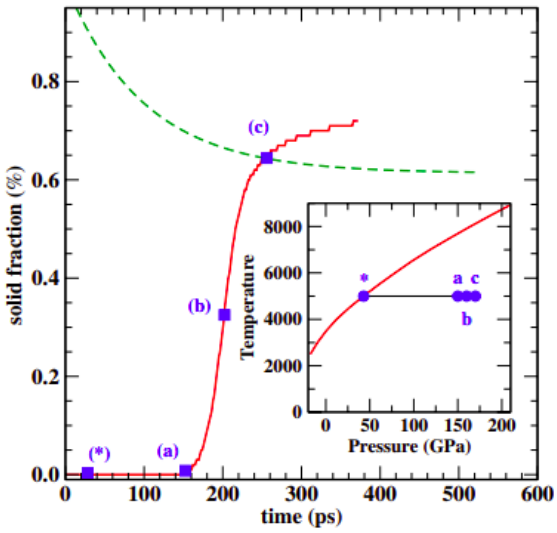

Figure 1.1: Graph shows the percentage of the system that solidified as time went on when pressure was exerted [2].

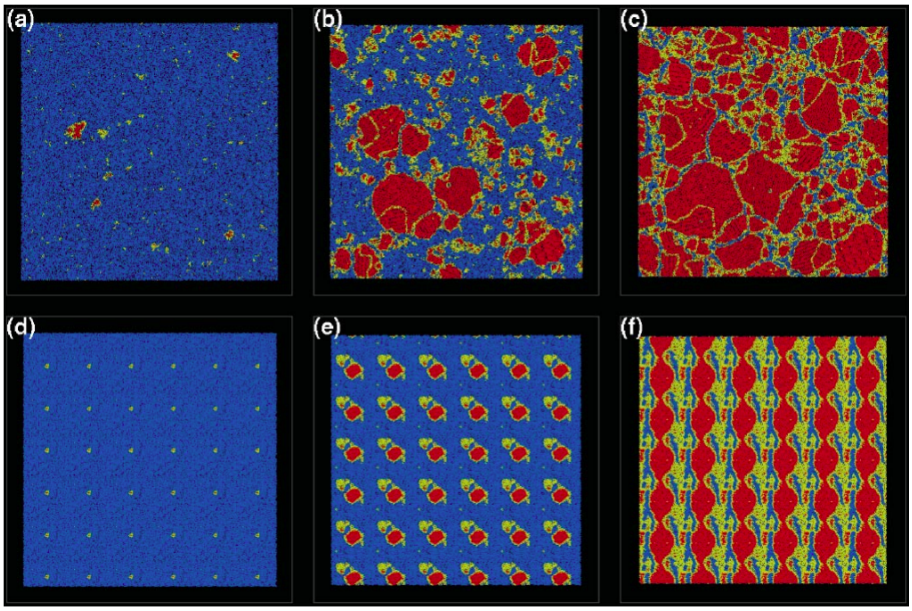

Figure 1.2: Sub-figures a-c show the pressure-induced nucleation of the 16M atoms tantalum system, and sub-figures d-f show the pressure-induced nucleation of the 64k atoms system, replicated 250 times in total across all three dimensions [2].

Yet another method for calculating the melting temperature involves a more theoretical approach. The Gibbs free energy of a system is calculated using ab-initio methods such as density functional theory (DFT)[3]. This so-called free-energy approach makes use of a reference system to reproduce the Gibbs free energy as a function of temperature of liquid and solid states, and finds the change in slope from solid to liquid to determine the enthalpy of fusion as shown in Figure 1.3. This enthalpy of fusion is used to determine the melting temperature T^m.

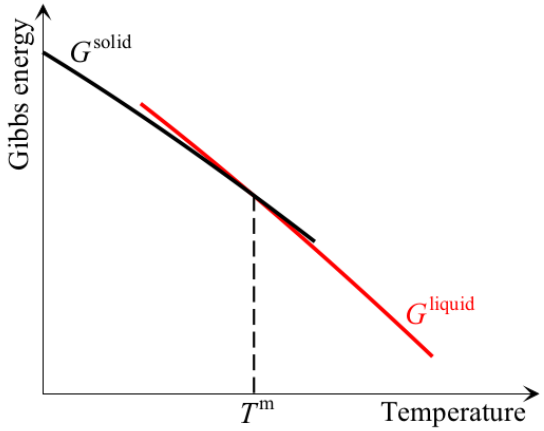

Figure 1.3: Free energy approach for determining the melting temperature T^m at constant pressure by finding the intersection of the solid state Gibbs free energy G^{solid} and the liquid state Gibbs free energy G^{liquid} [3].

The application of machine learning methods to these two other methods, even the one used in this work, is also possible but with more adjustments. In particular, one would need to break the plotting in smaller intervals of graphs such as those in Figures 1.2 and 1.3 for the identification of phase change by the neural networks used in this work and subsequent numerical computations.

Yet another method is the so-called two-phase coexistence, which required several more particles than the previous methods. In this method, portions of the system must exist in both solid and liquid phases as show in figure 1.4.

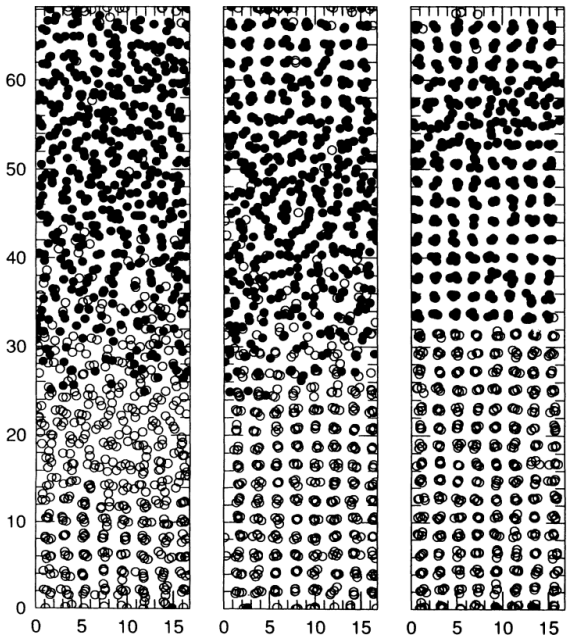

Figure 1.4: The two phase coexistence entails having both solid and liquid phases portions of the system [4]. Solid circles are atoms initially in the solid phase, while open circles are atoms initially in the liquid phase.

In summary, methods for mapping the melting line include the following:

- z-method
- pressure-induced nucleation
- Gibbs-based methods
- two-phase coexistence

1.1 Motivation

The purpose of this study stemmed from the more manual computation of data points for the melting line of copper (Cu) using the so-called z-method. Using this method requires a molecular dynamics (MD) simulation of the micro-canonical ensemble of a system at a

specified temperature for various temperatures and for sufficiently long simulation times. The temperature versus time graph needs to be plotted, and a sudden drop in temperature needs to occur to indicate melting. The accuracy of the melting temperature can be increased, but this still requires the same type of simulations and amount of simulation time but just at a shorter temperature range. The temperature range can continually be shrunk to increase the melting temperature accuracy. This is a very time consuming and tedious task as a user needs to plot the graphs and note where the melting happens, and the need for a sufficiently long simulation time, which in the more manual computation of Cu shown in this paper is 10ps which yields to about 10 minutes of real time for a parallel run in the MD software used with four processors, which leads to the user requiring a lot of patience.

No programs seem to exist online that facilitate this procedure, and, furthermore, no programs seem to exist that go beyond just finding the melting line of a metal like Cu, such as mapping the phase transitions between crystals or conveniently creating higher-dimensional phase diagrams (i.e. those beyond just two dimensions). These programs can be very useful for the analysis of systems and the testing of inter-atomic potentials in molecular dynamics.

The chapters of this paper are organized sequentially and information from previous chapter is referenced in later chapters. In this chapter we introduced our thesis and our motivation. In Chapter 2 we go over some thermodynamic and solid state concepts to better understand the type of systems we are analyzing. In Chapter 3 we cover molecular dynamics simulations and how we applied them to naively compute the melting line of copper. In Chapter 4 we introduced convolutional neural networks, the center-piece of the proposed method, along with a binary search algorithm and a shrinking enclosure algorithm, both necessary to aid and improve the efficacy of the neural network. In Chapter 5 we describe the implementation of the machine learning method and two algorithms introduced in Chapter 4, including the pseudo-code. In Chapter 6 we apply our proposed method to the same copper system along with two other new systems, FCC aluminum and BCC tantalum.

In chapter 7 we discuss the efficacy and shortcomings of our proposed method. Finally, in Chapter 8 we talk about future work, including promising algorithms and more intricate phase diagrams we may be able to map using machine learning methods.

Chapter 2

Systems of Interest

2.1 Thermodynamic Analysis of a System

2.1.1 Power of Processes Exerted on a System

A process in thermodynamics is defined as any interaction between the environment and the system, the system being the part of the universe that is of interest and analyzed and the environment being everything else. Processes are any interactions between the environment and the system. Three types of processes can be characterized: mechanical, thermal, and chemical. Looking at the thermodynamic analysis of a system as in book [5], chapter 1, these processes are given quantitative meaning respectively by work on or by the system, heat transfer, and chemical work on or by the system. Work on or by the system is the pressure p times the negative in change in volume ΔV of the system, which must be a deformation of the system's volume, defined by the system's boundaries. Heat transfer occurs when there is a temperature difference between the system and the environment, in which heat flows from higher to lower temperature. It is defined by the product of temperature T and a change in entropy ΔS. Chemical work is the diffusion of particles across the system boundaries. The system does chemical work when it gives up particles, and chemical work is done on the system when it receives particles from the environment. The chemical work done on the system involving particles of species (where species is a type of particle) A is given by the product of the chemical potential of A – denoted by μ_A times the change in particles of species A – denoted by ΔN_A.

Processes can change the value of a state variable (or more generally state functions

where a state variable is a trivial state function) of the system. State variables specify the state of the system entirely. Extensive state variables are the sum of state variables of compartmentalized parts of the system (subsystems). These have respective conjugated paired intensive variables, which do not posses the same property and thus are system-size independent. These conjugated pairs are partial derivatives of the internal energy of the system of each other as shown in Equation (2.1)

$$\frac{\partial E}{\partial X_i} = Y_i, \frac{\partial E}{\partial Y_i} = X_i \tag{2.1}$$

where E is the internal energy of the system, X_i is an extensive variable, and Y_i is its intensive variable pair.

For our purposes, we want to look at the extensive variables entropy S, volume V, and number N_A of particles of a certain species A in the system, and at their respective conjugated variables: temperature T, pressure p, and chemical potential μ_A of this species A.

It is worth explaining entropy as the natural logarithm of the degeneracy of a system at a particular energy, given by the following formula, chapter 2 of [13],

$$S(N, V, E) = k_B \ln(\Omega(N, V, E)) \tag{2.2}$$

Here, Ω is the degeneracy of the system at a certain state (N, V, E). The natural logarithm is taken to make entropy an extensive variable; k_B is the Boltzmann constant, and the system is fixed at a certain number of particles N, volume V, and internal energy E.

While most of the state variables mentioned are self explanatory, it also worth clarifying that the chemical potential of a species A, usually denoted as μ_A, is the change in the internal energy of the system when a particle of species A enters the system, and it would be the energy lost when a particle of species A leaves the system.

It is also worth noting that the temperature of the system is related to the total kinetic

energy of the system by the following [14]

$$dNkT = \sum_{i=1}^{N} m_i |u_i|^2 \tag{2.3}$$

Here, u_i is the velocity of particle i, m_i is its mass, d is the number of dimensions of the system (e.g. 2D or 3D) and k is a proportionality constant. We can use the state variables we are interested in to completely define the state of the system.

If we consider Equation (2.1), we can use the chain rule and write in general

$$\frac{dE}{dt} = \sum_i Y_i \frac{\partial X_i}{\partial t} \tag{2.4}$$

Using Equation (2.4), we can look at the powers exerted on the system and look at the rate of change of the internal energy with respect to time with Equation (2.5) shown below

$$\frac{dE}{dt} = T\frac{dS}{dt} - p\frac{dV}{dt} + \sum_A \mu_A \frac{dN_A}{dt} \tag{2.5}$$

The last term in Equation (2.5) is a summation over the number of different species A in the system.

It is worth noting that the term $p\frac{dV}{dt}$ in Equation (2.5) has a negative sign because a positive change in volume $\Delta V > 0$ indicates the system doing work on the environment, and thus losing energy. In engineering, the sign is positive because engineers are more interested in building machines that do work, but here we use the physics convention.

In literature, the internal energy E is derived from Equation (2.5) first using the so-called Gibbs-Duhem relation, which states that the sum of the terms involving the differentials of the intensive state variables T, p and μ_A equates to zero. This yields that the time integral of Equation (2.5) is just

$$E = TS - pV + \sum_A^S \mu_A N_A \tag{2.6}$$

This is the so-called Euler equation in thermodynamics. It will be of interest in future

sections.

2.1.2 Equipartition Theorem and Point Mass Particles with Harmonic Oscillator Potentials

The equipartition theorem states that the thermal energy of a system is distributed equally among all forms of energy. Or more generally and precisely, as shown in [15], given the Hamiltonian of a system H, which for our purposes will equal the energy of the system, we can consider all the homogeneous terms in the Hamiltonian as in Equation (2.7)

$$H = \sum_{l} g_l(x_i, .., x_j) + h \tag{2.7}$$

where g_l is some homogeneous term dependent on some subset, improper or not, of the components of the system's phase $\vec{x}$, as defined in Section 3.2, and h is independent of the phase coordinates in $\vec{x}$.

A function g has degree of homogeneity r if it satisfies Equation (2.8).

$$g(\lambda x_1, ..., \lambda x_L) = \lambda^r g(x_1, ..., x_L) \tag{2.8}$$

where λ is a constant. All such terms g_l in Equation (2.7) that have the same degree of homogeneity r and are dependent on the same number of arguments L make the same contribution to the mean total energy.

For our purposes, and as an example, we will look at point mass particles, i.e. matter particles that occupy no volume, that interact with each other via a quadratic potential U, which has the form $U = \frac{1}{2}k_c\Delta r$ with spring constant k_c and displacement Δr. Thus, no rotational kinetic energy terms are present and thus, for N particles, the Hamiltonian of the system is

$$H = \frac{1}{2}m_i \sum_{i}^{N}[u_{ix}^2 + u_{iy}^2 + u_{iz}^2] + \frac{1}{4}k_c \sum_{i,j}^{N}[(r_{ix} - r_{jx})^2 + (r_{iy} - r_{jy})^2 + (r_{iz} - r_{jz})^2] \tag{2.9}$$

Here, the sums are taken over the N particles, denoted by i or j. The last summation has an extra 1/2 factor to account for double counting for the sum over all the N i-particles and the N j-particles (a double summation where terms are repeated), and isotropy for the spring constant k_c is assumed.

According to the equipartition theorem, the steady-state energy contribution of each term is

$$g_L = \frac{1}{r}kT \tag{2.10}$$

where T is the temperature of the system.

If we split all the terms in square brackets in Equation (2.9) into functions of degree of homogeneity 2 that take 3 phase components as arguments, e.g. $\sum_i^N m_i u_{ix}^2$, and taking into account that $m_i u_{ix}^2 = p_{ix}/m_i$, then we have for each of the 3N homogeneous functions of degree 2

$$g_3 = \frac{1}{2}kT \tag{2.11}$$

We can rewrite the Hamiltonian in the more simpler form

$$H = \frac{1}{2}\sum_i^N \frac{\vec{p_i}^2}{m_i} + \frac{1}{4}k_c \sum_{i,j}^N r_{ij}^2 \tag{2.12}$$

where the dot product of $\vec{p_i}$ is taken with itself and where

$$r_{ij}^2 = (r_{ix} - r_{jx})^2 + (r_{iy} - r_{jy})^2 + (r_{iz} - r_{jz})^2 \tag{2.13}$$

With this notation, we split the Hamiltonian terms into the kinetic and quadratic potential energy contributions, and thus the internal energy E at the steady state is given by

$$E = \frac{1}{2}Ng_3 + \frac{1}{4}\cdot 2Ng_3 \tag{2.14}$$

In Equation (2.14), the first term come from the translation kinetic energy of the particles, and the second term comes from the vibrations of the particles. As can be seen, both

terms equal $\frac{1}{2} \cdot N \cdot g_3$, indicating equal contribution to the internal energy, or conversely, given power exerted on the system, in the steady state, the change in internal energy ΔE would split evenly between the velocity of the particles and the stretching of the "springs" (vibrational modes) between particles.

2.1.3 Virial Theorem

The virial theorem is related to the aforementioned equipartition theorem. The virial theorem applies to a system of particles if the position and momentum of each particle is bounded, which is the case for a solid crystal system. As discussed in the section on the virial theorem in [16], the virial theorem can be derived by starting with Newton's second law which on a single particle a is

$$\vec{F_a} = m_a \ddot{\vec{r}} = \dot{\vec{p_a}} \tag{2.15}$$

where the superior dots indicate time derivatives. This vector equation can be multiplied by $\vec{r_a}$ for the scalar product and produce

$$\vec{r_a} \cdot \vec{F_a} = \vec{r_a} \cdot \dot{\vec{p_a}} = \frac{d(\vec{r_a} \cdot \vec{p_a})}{dt} - m_a \cdot \dot{\vec{r_a}}^2 \tag{2.16}$$

Noting that the kinetic energy of atom a is $\frac{1}{2} \cdot m_a \cdot \dot{\vec{r_a}}^2 = T_a$, we can write

$$\frac{d(\vec{r_a} \cdot \vec{p_a})}{dt} = \vec{r_a} \cdot \vec{F_a} + 2T_a \tag{2.17}$$

The time average of a function f is

$$\bar{f} = \frac{1}{\tau} \int_0^\tau f dt \tag{2.18}$$

Applying this to Equation (2.17) we get

$$\frac{1}{\tau}[\vec{r_a} \cdot \vec{p_a}]_0^\tau = \overline{\vec{r_a} \cdot \vec{F_a}} + 2\bar{T}_a \tag{2.19}$$

where the superior bar indicates the time average.

The left-hand-side of Equation (2.19) can be made 0 if the motion is periodic or the period for averaging τ can be made large enough to make the left-hand-side be close enough to 0. Doing this we arrive at

$$-2\bar{T}_a = \overline{\vec{r_a} \cdot \vec{F_a}} \tag{2.20}$$

If we have a system of more than one particle, we can just sum over all the particles and obtain

$$-2\bar{T} = \sum_a \overline{\vec{r_a} \cdot \vec{F_a}} \tag{2.21}$$

The right-hand-side of Equation (2.21) is called the *virial*.

Now, non-conservative forces (e.g., friction) on particles time-average to zero. Conservative forces are derived from a potential. For a conservative force on atom a from a potential U, the following follows

$$\vec{F_a} = -\nabla_a U \tag{2.22}$$

Here, the gradient is taken with respect to the position coordinates of particle a since the potential would involve arguments at least from the position of some other particle.

If the potential is a homogeneous function of degree r as defined by Equation (2.8), then making the replacement of Equation (2.22) into right-hand-side of Equation (2.21) yields

$$-\sum \vec{r_a} \cdot \vec{F_a} = \sum \vec{r_a} \cdot \nabla_a U = rU \tag{2.23}$$

where, as mentioned earlier, r is the degree of homogeneity of U. With this, we can finally write

$$2\bar{T} = r\bar{U} \tag{2.24}$$

If U is approximated by a quadratic potential, then U has a degree of homogeneity of 2 and thus we arrive at

$$\bar{T} = \bar{U} \tag{2.25}$$

Equation (2.25), along with section 2.2.3 will, be used to justify the initial velocity re-scaling of particles in a Cu system for finding the melting point at a given pressure.

2.2 Crystal Systems

2.2.1 Types of Crystal Structures

Crystal structures, as opposed to amorphous materials, have a repeating pattern across space, determined by a lattice and a basis, where the lattice is all geometric points in space, with position $\vec{r'}$ determined by the lattice translation vectors $\{\vec{a_i}\}$ [17]

$$\vec{r'} = \vec{r} + u_1\vec{a_1} + u_2\vec{a_2} + u_3\vec{a_3} \tag{2.26}$$

and the basis is all groups of particles attached to a lattice point. The position of each particle j is given by

$$\vec{r_j} = x_j\vec{a_1} + y_j\vec{a_2} + z_j\vec{a_3} \tag{2.27}$$

The factors x_j, y_j and z_j are between 0 and 1, and thus all particles reside within the conventional unit cell delimited by the $\{\vec{a_i}\}$ with volume V_c

$$V_c = |\vec{a_1} \cdot (\vec{a_2} \times \vec{a_3})| \tag{2.28}$$

Fourteen types of lattices exist in three dimensions, defined by point group symmetric operations (i.e. operations that move all but one point across space but keep the geometry of the lattice the same) or the number and type of constrains in the cells. The lattice types are divided into six groups as shown in Table 2.1 (pg. 9 of [17])

Table 2.1: The 14 lattice types in three dimensions.

System	Number of lattices	Restrictions on conventional cell axes and angles
Triclinic	1	$a_1 \neq a_2 \neq a_3$ and $\alpha \neq \beta \neq \gamma$
Monoclinic	2	$a_1 \neq a_2 \neq a_3$ and $\alpha = \beta = 90° \neq \gamma$
Orthorhombic	4	$a_1 \neq a_2 \neq a_3$ and $\alpha = \beta = \gamma = 90°$
Tetragonal	2	$a_1 = a_2 \neq a_3$ and $\alpha = \beta = \gamma = 90°$
Cubic	3	$a_1 = a_2 = a_3$ and $\alpha = \beta = \gamma = 90°$
Trigonal	1	$a_1 = a_2 = a_3$ and $\alpha = \beta = \gamma < 120°, \neq 90°$
Hexagonal	1	$a_1 = a_2 \neq a_3$ and $\alpha = \beta = 90°, \gamma = 120°$

All these lattice types of parallelepiped figures (i.e., three dimensional figures composed of 6 parallelograms joined at the edges) need different types of constraints on the three side lengths and three angles to completely define them geometrically. For our purposes, we are interested in the cubic system, which consists of the simple cubic (SC), body-centered cubic (BCC) and face centered cubic (FCC) lattices. Shown in Figure 2.1 are the conventional unit cells of these systems generated with the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) software [18].

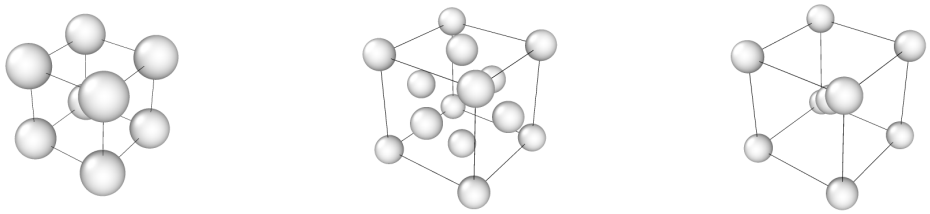

Figure 2.1: Simple cubic, face centered cubic and body centered cubic conventional unit cells from left to right.

As shown in Table 2.1 for the cubic system, there are 5 constraints on the cubic system, meaning only the side length of the cube is needed to define the lattice types in this system. We call this side length the lattice constant.

2.2.2 Types of Phase Transitions

As discussed in Chapter 6 of [5], pg. 140, "A phase is a state of matter which occupies a subspace of the state space characterized by physical properties of the system." As discussed previously, examples of state variables that can make up a state space include volume V, entropy S, energy E, etc. Examples of phases include solid, liquid, gas, plasma, ferromagnetic, superconducting and superfluid. In addition, more nuance types of phases in a solid can be the different crystal structures in Table 2.1.

A system can transition from one phase to another. Examples of these transitions are melting, solidification, vaporisation, condensation, sublimation and deposition. Phase transitions also include transitions from one crystal structure to another, such as from iron FCC to iron BCC.

One thing to clarify is that the phase spoken of here is that related to solid state physics. The previously mentioned phase of a system denoted by the vector $\vec{x}$ is that related to molecular dynamics. The difference should be clear by the context.

Certain state functions change when there is a phase transition. Phase transitions occur in response to physical processes. For example, when heat is added, a solid can melt, the temperature stays constant but the energy increases because there is a release in the inter-particle bonds. More specifically, as mentioned in pg. 140 of [5], according to the Ehrenfest classification, there are two types of phase transitions.

First-order phase transitions are characterized by discontinuities in the partial derivatives of the Gibbs free energy with respect to volume V and entropy S. The Gibbs free energy is a Legendre transformation with respect to the entropy and volume. Normally, the internal energy is a state function of the extensive quantities entropy S, volume V and number N_A of particles of the species A. However, in experiments, it is many times more practical to control some of the intensive quantities (i.e., temperature T, pressure p and chemical potential μ_A). The Gibbs free energy gives all the same information as the internal energy, and one can be converted into the other by a Legendre transformation. This type of transformation replaces a variable by the partial derivative of the function in question with

respect of that variable. The extensive-intensive conjugate pair relationship of Equation (2.1) makes it such that for internal energy, extensive state variables are replaced by their intensive conjugate pairs and vice versa. The Gibbs free energy is defined as below

$$G(T, p, N_A) = E + pV - TS \tag{2.29}$$

Here, the expression in the right is computed and then the variables V and S are replaced by their conjugates p and T in the equation through their relationships $p = \dfrac{\partial E}{\partial V}$ and $T = \dfrac{\partial E}{\partial S}$, for whatever form the state function E has.

An example of the discontinuity mentioned is shown in the temperature versus Gibbs free energy graph in Figure 2.2.

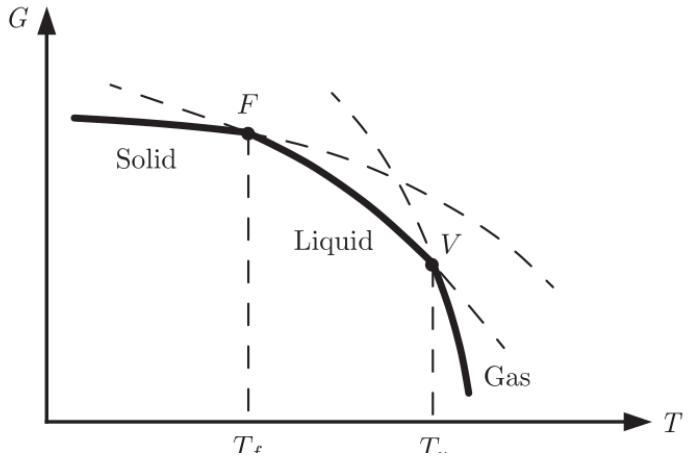

Figure 2.2: Temperature T versus Gibbs free energy G graph. The partial derivative of the Gibbs free energy with respect to temperature is the entropy, and discontinuity in this can be seen on the graph by sharp turns of the graph ([5] pg. 141)

In this paper, we will focus mainly on phase transition of the first order, but it is worth mentioning that phase-transitions of the second order are characterized by discontinuities in the second partial derivative of the Gibbs free energy. An example of this is when the state of a system evolves through the critical point, which is the point where two phases of

matter become indistinguishable from each other, such as at high pressure and temperature liquid water and water vapor.

2.2.3 Quasi-Harmonic Approximation of Inter-Atomic Energies

Here, we want to show that we can approximate the interaction between two particles in a solid crystal by a harmonic oscillator. As our test case, we look at two Cu atoms, whose interaction is dictated by the potential developed by Mishin et al. in 2001 [19], which is an embedded atom method (EAM) type of potential. We want to approximate this by a two mass spring system and determine the frequency of oscillation between the two Cu atoms.

Simulating a system of two Cu atoms, initially separated by 3 Angstroms, starting at 0 temperature, we plot the separation distance as a function of the pairwise energy U and after we approximate around the minimum with a quadratic polynomial as shown in Figure 2.3.

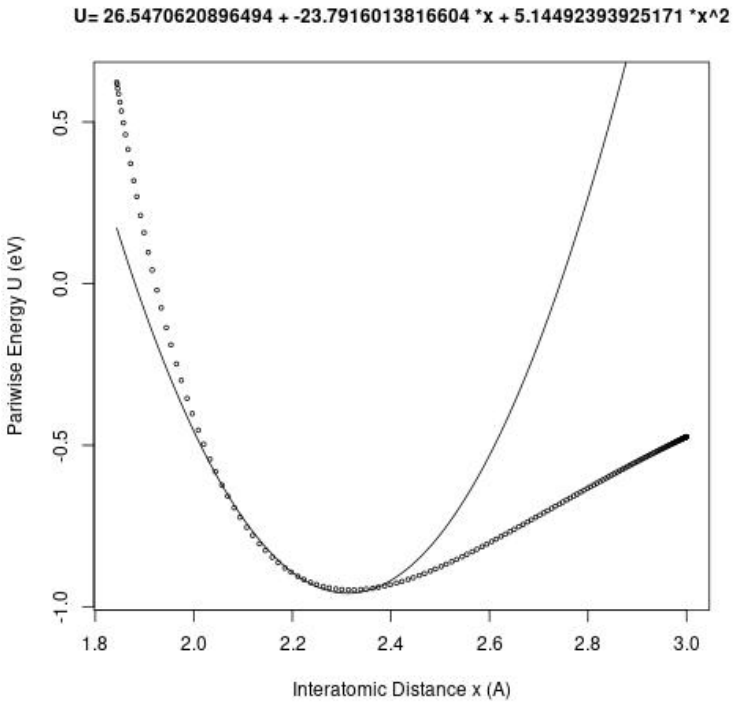

Figure 2.3: Quasi-harmonic interaction between two isolated copper atoms approximated by a quadratic polynomial.

Here, data points with inter-atomic distance values in the range $[2.01363, 2.41363]$A are included, which includes the minimum (2.31363A, -0.947211eV) of U, which are the equilibrium distance and the minimum energy respectively. The formula for the quadratic interpolation is shown at the top of the graph.

Another system where a similar analysis can be made is on the Cu face centered cubic crystal in which we look at only two Cu atoms, whose pairwise energy graph and interpolate quadratic polynomial are shown in Figure 2.4. We simulate this system in MD, do the same kind of interpolation and apply the same analysis. Now, we want to approximate U with a truncated Taylor expansion up to the square term about the minimum and equate to the quadratic interpolation formula we have.

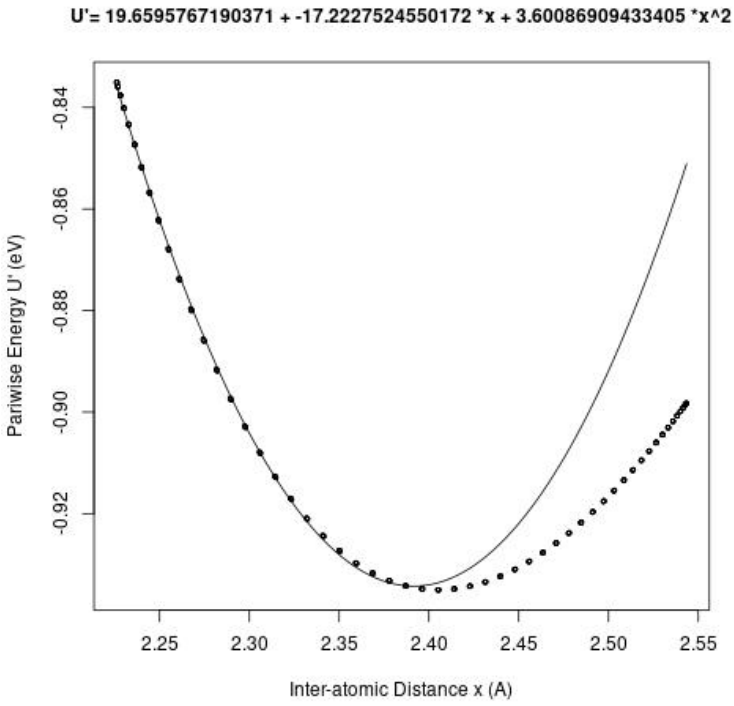

Figure 2.4: Quasi-harmonic interaction between two copper atoms in an FCC crystal approximated by a quadratic polynomial.

The truncated Taylor expansion about the equilibrium distance x_0:

$$U(x) = U(x_0) + \frac{\partial U}{\partial x}(x_0)(x - x_0) + \frac{1}{2}\frac{\partial^2 U}{\partial x^2}(x_0)(x - x_0)^2 \tag{2.30}$$

If we approximate about the minimum, then by definition, the first derivative $\frac{\partial U}{\partial x}(x_0) = 0$. Thus, we are left only with the constant and square terms:

$$U(x) = U(x_0) + \frac{1}{2}\frac{\partial^2 U}{\partial x^2}(x_0)(x - x_0)^2 \tag{2.31}$$

As such, we want our interpolated polynomial to resemble this form, so we complete the

square using the following formula:

$$c + bx + ax^2 = c - \frac{b^2}{4a} + a\left(\frac{b}{2a} + x\right)^2 \tag{2.32}$$

We can equate the quadratic interpolation formula to our truncated Taylor and see that:

$$\frac{1}{2}\frac{\partial^2 U}{\partial^2 x}(x_0) = a \tag{2.33}$$

Now we want to recall the model of a two spring system to determine the frequency and period of oscillations between the two Cu atoms. The model of a two mass spring system is governed by this equation of motion:

$$\mu\frac{d^2x}{dt^2} = -kx \tag{2.34}$$

Here, μ is the reduced mass of the two masses m_1 and m_2, given by $\mu = \dfrac{m_1 m_2}{m_1 + m_2}$, and k is the spring constant. Reformulating this differential equation in the following form gives:

$$\frac{d^2x}{dt^2} + \frac{kx}{\mu} = 0 \tag{2.35}$$

This differential equation is solved by a sinusoidal of amplitude A, phase ϕ and angular frequency ω: $A\sin(\omega x + \phi)$. A and ϕ are determined by the initial conditions, but the angular frequency ω is determined by the spring constant k and reduced mass μ by:

$$\omega = \sqrt{\frac{k}{\mu}} \tag{2.36}$$

The reduced mass model of the two mass spring system treats the original system of two masses as one. Thus, we can use the potential energy of the single mass attached to a spring on this reduced mass model. Recall that the potential energy of a mass in a spring

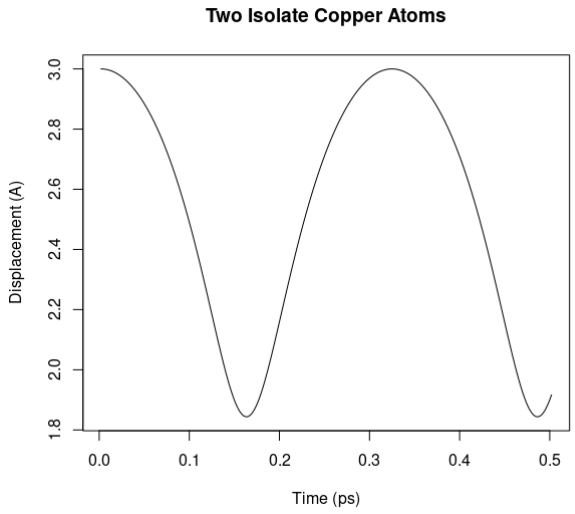

Figure 2.5: Simulation data of two isolated Cu atoms of time versus displacement.

is given by

$$U = \frac{1}{2}k(x - x_0)^2 \tag{2.37}$$

This analysis is not perfect since the interpolating polynomial needs to include a certain range of data points such that the period of oscillation in the simulation is consistent with the value of a in Equation (2.32). For example, figure 2.5 shows the displacement versus time of the two isolated Cu atoms system. The value of a needs to be consistent, in accordance with Equations (2.33), (2.36), and (2.37), with the period in Figure 2.5, which in this case is about 0.32 ps.

This analysis is just to show that the interaction between two Cu atoms, part of an FCC crystal or isolated, can be considered quasi-harmonic (i.e. almost harmonic), and thus be approximated by a harmonic potential within a certain range of the displacement from equilibrium between particles. This, along with Subsections 2.1.2 and 2.1.3, will tie in with the way we need to initialize the temperature of our systems to attain a desired temperature for our determination of melting lines using the z-method.

Chapter 3

Molecular Dynamics

3.1 Newton's Equations of Motion

Molecular dynamics (MD) is a mostly classical view, as opposed to taking quantum mechanics into consideration, of the motion of particles. As described in [20], MD can be described as computational statistical mechanics, where give the rules for the system, MD can be used to quantify system properties, such as temperature. A system is considered composed of discrete particles and an equation of motion like Equation (2.15) is sought. The application of Newton's second law, in the most basic implementation of MD, leads to the time evolution of the system. The force F_a on a particle a is derived from some potential as in Equation (2.22). The evolution of the positions of the particles is sought from the equations of motion. These particles' trajectories require an integration method that is numerical in nature, for the particles' equations of motion are coupled since the potential U in Equation (2.22) always involves the phase coordinates of more than one particle, since potential energy is an energy existing only through interactions. It is well known that there is no analytical solution for a three or more body problem when there is interaction between the particles, leaving numerical solutions as the only option.

Aside from Newton's second law, more powerful methods of determining the system's particles' causation of motion are through Lagrangian and Hamiltonian dynamics. As shown in Chapter 3 of [21], the equations of motion for Lagrangian dynamics with constraints (for more generality) are

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \sum_l \lambda_l a_{li},\ i = 1, ..., N \tag{3.1}$$

where $\{q_i, \dot{q}_i\}$ are the generalized coordinates and velocities, a_{li} involve coefficients for linear constraints on the generalized velocities, and the Lagrangian L is

$$L = T - U \tag{3.2}$$

where T is the total kinetic energy, U is the total potential energy, and $L = L(\{q_i\}, \{\dot{q}_i\}, t)$.

The Hamiltonian equations is a Legendre transform of the Lagrangian as shown

$$H = \sum_i \dot{q}_i p_i - L \tag{3.3}$$

where the $\{p_i\}$ are the generalized momenta. The Hamiltonian equations of motion are given by

$$\dot{q}_i = \frac{\partial H}{\partial p_i}, \ \dot{p}_i = -\frac{\partial H}{\partial q_i} \tag{3.4}$$

For our purposes, equations of the form of (2.22), which derive from Equation (3.1) if the $\{q_i\}$ correspond to the Cartesian coordinates, for each particle of the system, will suffice, but as previously shown in Subsection 2.1.2, it is useful to make use of the Hamiltonian operator.

For the MD software used in this work, LAMMPS, we will make use of the Verlet integration algorithm, which is generally the case by default for time-stepping (i.e. progression of time in the simulation) in LAMMPS simulations. This algorithm, as shown in chapter 2 of [21], is shown below in Equations (3.6), (3.7), and (3.5).

Taking h as the length of the time-step (i.e., the Δt since this is a numerical method partitioning the time integral into time intervals of size h), we first update the particles' velocities by half a time-step, getting $a_{ix}(t)$ from the current force, which is taken from Equation (2.22):

$$u_{ix}(t + h/2) = u_{ix}(t) + (h/2)a_{ix}(t) \tag{3.5}$$

Afterwards, we advance the positions a full time-step using the half-time-step advanced

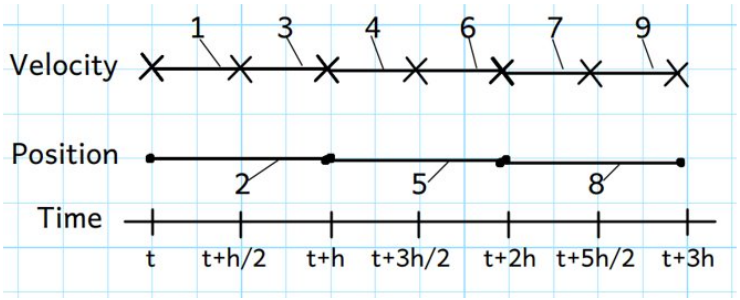

Figure 3.1: As can be seen, the velocity is updated half a time-step with the current positions for the acceleration, the positions follow for a full time-step, and finally the velocities follow again for another half time-step using the just updated positions for the acceleration.

velocities from Equation (3.5):

$$r_{ix}(t+h) = r_{ix}(t) + hu_{ix}(t+h/2) \tag{3.6}$$

Lastly, we use the full-time-step advanced positions from Equation (3.6) to compute the potential, then the forces, and finally the needed accelerations $a_{ix}(t+h)$:

$$u_{ix}(t+h) = u_i x(t+h/2) + (h/2)a_{ix}(t+h) \tag{3.7}$$

The Verlet method can be seen visually in figure 3.1.

This three-step process is repeated until the simulated time-length is reached. The user usually specifies the number of time-steps for the desired simulated time. The advantage of Verlet is that it is a relatively simple and computationally inexpensive time-integration method (e.g. as compared to high order Runge-Kutta methods), yet it has smaller-order errors. Verlet has errors of order $O(h^4)$ for the coordinates and $O(h^2)$ for the velocities, as compared to the Euler method which has an error of order $O(h)$ for the coordinates.

3.2 Types of Ensembles

An ensemble is, in theory, an infinite set of versions of a system, where each version has the same values for particular state variables but not the same individual micro-states for each individual particle of the system. We can define the phase of a system of N particles as a vector $\vec{x}$, just like it was done in [22], which is the concatenation of all the system particles' positions r_i and momenta p_i, where $p_i = m_i u_i$. The ensemble consists of infinite versions of the same system because it aims to include all possible phases of the system that fall under the specified constraints.

As such, we say that the systems are in the same state, depending on which state variables we want all the versions to have the same, but in different phases. Depending on which state variables we want to keep constant, we determine a different type of ensemble, as described in [13].

The micro-canonical ensemble is that in which all the systems in the ensemble have the same energy. This is also denoted as the NVE ensemble for these are the state variables that we keep constant. In terms of Equation (2.5), this means that $\frac{dN_A}{dt} = 0$ for all A, $\frac{dV}{dt} = 0$ and $\frac{dE}{dt} = 0$. Another kind of ensemble is the canonical ensemble, also denoted as the NVT ensemble. In this ensemble, we keep the temperature constant (i.e. $\frac{dT}{dt} = 0$) instead of the energy. Finally, we can also consider the grand-canonical ensembles, also denoted as the μVT ensembles. In this ensemble, as compared to the canonical ensemble, the chemical potential is held constant (i.e., $\frac{\mu_A}{dt} = 0$) instead of the number of particles. This type of ensemble can represent systems where there is both heat and chemical work processes.

The equations of motion for these ensembles can be summarized as follows:

1. NVE: use of the standard Newton's equation of motion $F = ma$

2. NVT: thermostats to keep T constant that make use of extended variables such as Langevin [23], Berendsen [24], velocity re-scaling, Nose-Hoover [25], etc.

3. μVT: similar to the NVT ensemble but $\frac{\mu_A}{dt} = 0$

As discussed in [26], all thermodynamics properties of a system can be computed from any of the types of ensembles. Here we mention only three, but any ensemble can be define by the set of constant-kept state variables. For our purposes of finding the melting line of various metals, we will focus on the micro-canonical ensemble to compute the melting temperature at a constant pressure. But for future applications (Sections 8.2 and 8.3), other types of ensembles may be considered.

3.3 Normal Method for Determining Copper Melting Line

The type of phase transition we will focus on is melting, as the title suggests, and the method that we will use is the so-called z-method. As described in [6], in this method, a user runs a simulation in the NVE ensemble at a certain temperature, and spontaneous melting occurs if the temperature is high enough. When melting occurs, there is an increase in the latent heat of fusion. This increase in potential energy leads to a decrease in temperature. The method is called the z-method because the value from which the temperature falls from when melting occurs is a super-heating temperature, and the temperature the system falls to is the melting temperature. Plotting the temperature versus energy graph leads to a z shape as shown in the left graph of Figure 3.2. Repeating this procedure for different volumes can be used to map the melting line of a metal as shown in the right graph of Figure 3.2.

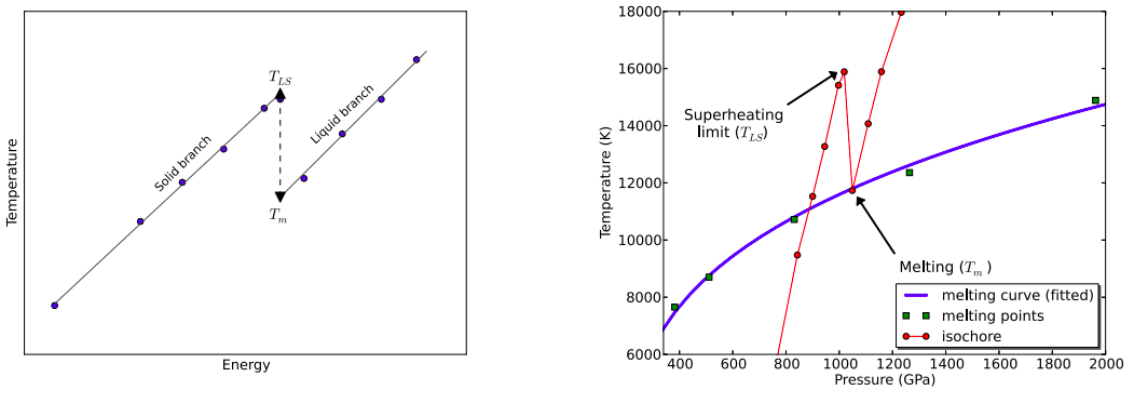

Figure 3.2: In the left graph we see the drop from the super-heating temperature to the melting temperature. On the right graph, we can see how this can be used to map the melting line in the P-V graphs. P-V points on the left side of the melting line correspond to solid states, and P-V points ont the right-side correspond to liquid states[6].

To determine the melting line of FCC copper, we used the so-called z-method. This method is describe as follows.

To determine the melting temperatures of FCC Cu at various pressures, we ran a micro-canonical ensemble simulation of a system of 5324 Cu atoms. We controlled the volume of the system by specifying a lattice constant for the FCC conventional unit cells. In turn, the system's pressure was controlled by the volume since the lattice constants of the FCC cells at 0 GPa must be about 3.6A. Thus, a smaller lattice constant indicates that the system is under compression, i.e. positive pressure. In this manner, we determined the pressures at which the system melts after determining the temperatures at which it melted.

The simulations ran for $2 \cdot 10^4$ steps with a step size of 1 femto-second (fs), leading to 20 picoseconds (ps) of simulated time.

We first sought a range to where the MD software program would reasonably work and to where the pressure that the system melts would reach the 200 GPa range. The range in lattice constants that worked was from 3.15 to 3.50 Angstroms. Starting from the lower bound of this range, we sought a lattice constant every 0.035A, leading to 11 lattice

constants. The way the system was set to a certain temperature was through velocity re-scaling. In theory, this would mean that the velocities of the Cu atoms would have to have magnitudes so that in Equation (2.3) the temperature T was the desired value. However, because of the equipartition theorem and virial theorem of Subsections 2.1.2 and 2.1.3, the temperature value must be double for the initial temperature as shown in Equation (3.8).

$$T_{needed} = 2T_{wanted}. \tag{3.8}$$

This is because half of the kinetic energy will go to the modes of the springs, i.e. the potential energy terms of Equation (2.9), and the other half will stay with the atoms' velocities. Therefore, the temperature in the MD software must be set to twice the desired temperature value in Equation (2.3). It is important to emphasize that this is for an initial run, where no previous phase of a previous simulation is used to initialize the system. This is done and explained shortly.

The temperature value that yielded the first time the system melted was determined by plotting the temperature against the time. Melting occurred when the temperature had a sudden drop. The pressure was noted at this temperature.

The reason this temperature drop and pressure increase happen is because we are running an NVE simulation. In this simulation type, the state variables N, V and E are kept constant. In addition, we need look at Equation (2.6). Because the system is melting, the entropy S must increase. Therefore, if we look at Equation (2.6), in order for the internal energy E to stay constant, the other free state variables T and p must change as well to compensate.

A time-average of both the temperature and the pressure for the last 15% of the simulation time was used as the recorded values for the melting temperature and pressure. It is important to note that these are the values after the system melts, and the average is taken to cancel out the natural fluctuations. The change in the system for both state variables can be seen in Figure 3.3. As can be seen, whereas the temperature has a sudden

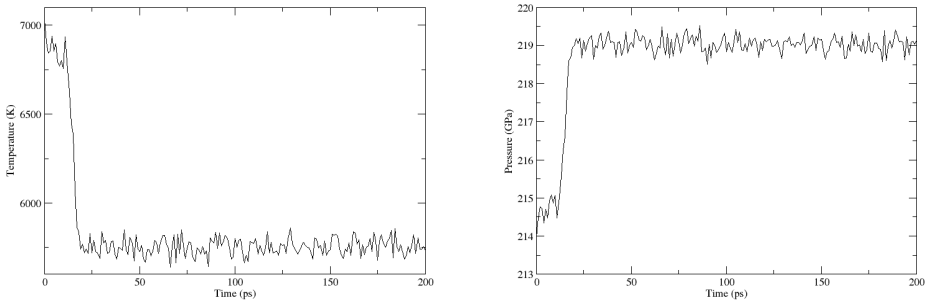

Figure 3.3: Change in temperature and pressure due to melting for a system of Cu atoms in FCC with conventional unit cell lattice constant of 3.150A.

decrease, the pressure has a sudden increase. This melting could occur at any time in the simulation. This is part of the reason why the simulation times were so long (20ps), to capture the moment the system melted if it was meant to melt at that temperature.

We did enclosures for where the melting point would be found, starting from 3200 to 7200 K at steps of 500 K for the given lattice constant set. We found the temperature from this discrete search and repeated for each lattice constant for 100 K steps within a 500 K range, then 20 K steps for within a 100 K range, then 4 K steps for within a 20 K range, and finally a 1 K step for within a 4 K range. It is necessary to use the phase of the previous simulation to start the next simulation in the temperature set for each temperature range. To do this, a data file with the phase is dumped and used to initialize the next simulation. However, in the next simulation, the initial velocities are re-scaled by equating the temperature value T to the previous temperature plus twice the difference between this value and the desired temperature as shown in Equation (3.9).

$$T_{needed} = T_{previous\_wanted} + 2(T_{wanted} - T_{previous\_wanted}) \tag{3.9}$$

The temperature needed T_{needed} is the value set in the velocity-re-scaling command in the MD software that takes in the previous simulation's phase. The reason for this has the same

logic as that of Equation (3.8). The "springs" (the inter-atomic forces) are already stretched to the amount needed for the system to have the phase for the previous temperature. As such, one just needs the velocity portion normal temperature of Equation (2.3) plus the difference between temperatures for the modes of the springs (Equation (2.37)) in order to satisfy Equation (2.24). It is again important to emphasize that the velocity re-scaling used for the system with the very first temperature for each temperature set follows Equation (3.8) while each subsequent simulated system for the rest of the temperatures follows equation 3.9.

The method of partitioning the initial temperature range into ever-smaller ranges greatly reduced computation time as compared to the estimated time of 1 K steps for all lattice constants by a factor of about 5 to 6. Each simulation took about 12 minutes, and the number of simulations for each of the 11 data points in Table 3.1 was 29, yielding a total of 319 simulations and an estimated time of 66 hours for 4 processors running LAMMPS in parallel. This, however, does not take into account the need for the user to manually set the temperature range of the aforementioned cycles by looking at graphs that are like the left graph of Figure 3.3. This is a very tedious and exhausting task for the sets of simulations take hours to finish, and the new simulations must be started after the temperature range determination.

3.4 Melting Line of FCC Copper

3.4.1 Result of MD Simulation for Copper Melting Line

Here, we present the results of the naive implementation of the z-method as described in the previous section for determining the melting line of FCC Cu. By naive implementation, we mean linear search as described in Subsection 4.3.2 and with no other algorithmic or machine learning implementations. This procedure will be referred to as the "normal procedure". In addition, by the melting line we mean the curve marking the transition

Table 3.1: Normal procedure simulation results for the melting line of Cu.

Latt. Const. (A)	Melt. Press. (GPa)	Melt. Press. Err. (GPa)	Melt. Temp. (K)	Melt. Temp. Err. (K)
3.150	218.93	0.22240	5769.84	55.4534
3.185	189.03	0.19585	5356.34	52.1689
3.220	163.02	0.19153	4945.84	45.9514
3.255	140.26	0.16355	4611.78	39.8171
3.290	120.10	0.14182	4239.91	37.0017
3.325	103.14	0.11902	3984.07	29.7303
3.360	87.45	0.13422	3637.64	31.9107
3.395	74.11	0.12831	3397.62	28.4171
3.430	62.27	0.10306	3135.67	24.7346
3.465	51.44	0.11033	2836.19	25.5035
3.500	42.40	0.09805	2624.67	21.7108
3.600	21.67	0.06130	2050.37	12.7977

from solid to liquid in the temperature T versus pressure P graph (a two-dimensional phase diagram). The solid and melting states can be seen in figure 3.4. Since the lattice constant determines the pressure on the system, it is also included.

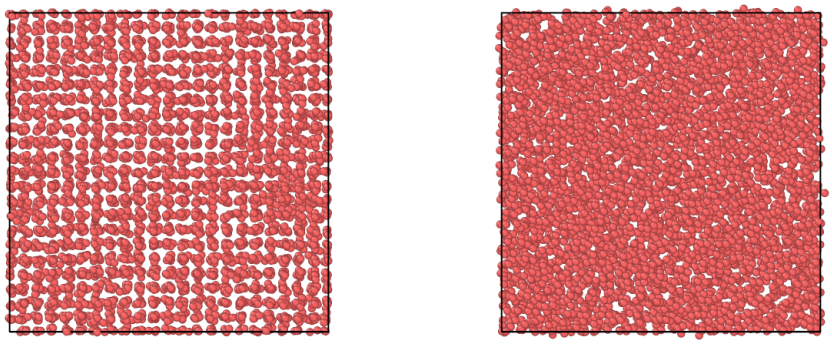

Figure 3.4: On the image in the left, the system of Cu atoms is vibrating but still solid FCC. In contrast, the system has melted in the image on the right.

3.4.2 Experimental Data of Copper Melting Line

Here, we present experimental data of the melting of Cu in figure 3.5. This data was taken from [27].

Table 3.2: Experimental data for Cu melting.

Press. (GPa)	Melt. Temp. (K)	Error in Melt. Temp. (K)
0.0	1354	5
4.0	1533	13
6.9	1628	18
9.8	1753	23
12.8	1803	15
12.8	1823	50
16.0	1923	15

3.4.3 Simon-Glatzel Equation Interpolation of Simulation and Experimental Data and Comparison

Finally, we present the data plotted along with the Simon-Glatzel interpolation from [28]. This interpolation equation tries to relate the melting temperature to pressure of a solid. It fits well for metals such as copper. The equation has the form

$$T_m = T_{ref}\left(\frac{(P - P_{ref})}{a} + 1\right)^b \tag{3.10}$$

Here, T_{ref} and P_{ref} is any data point. The one taken for each interpolation on this work is that of the melting temperature at zero pressure, thus $P_{ref} = 0$. All fits were done using the plotting tool xmgrace.

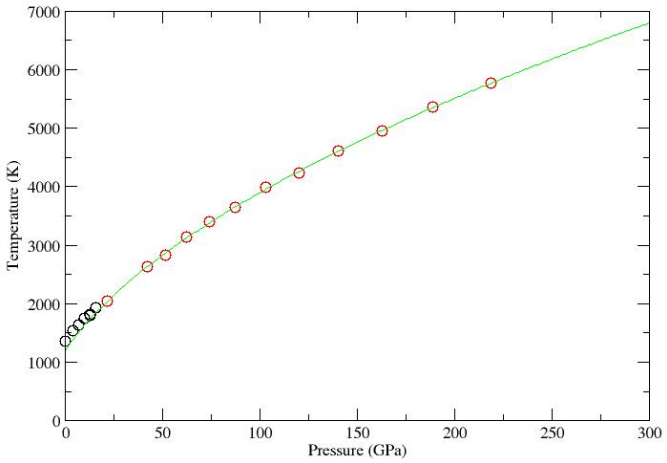

Figure 3.5: For FCC Cu system, experimental data (black circles), MD simulation data (red circles), and Simon-Glatzel fitting of MD simulation data (green line) plotted.

As can be seen, the Simon-Glatzel fitting from the MD simulation data coincides fairly well with the experimental data. For this fit, the parameters obtained were $T_{ref} = 1190.83$, $a = 12.8919$ and $b = 0.546141$. The experimental value of $T_{ref} = 1354$, as can be seen in Table 3.2. This leads to a relative percent error of 12.05% for the melting temperature at zero pressure.

Chapter 4

Convolutional Neural Networks in Machine Learning

In this section, we will go over the machine learning method that will be used to partially automate the determination of the melting line of Sections 3.4 and 3.5.

4.1 Basic Mechanism Behind Neural Networks and Applications

Neural Networks (NN) are a machine learning (ML) method that aims to mimic the in fact not completely understood mechanism of biological neural networks in the brain. As the name suggests, NNs are composed of neurons belonging to a layer. Neurons connect with each other to pass information. Neurons in the same layer are not connected, but neurons in different layers can connect in a multitude of ways. The simplest case is the so-called feed-forward NN where each neuron in a layer, except the input layer, gets input from each neuron in the previous layer. An example of such NN can be seen in Figure 4.1. This is the type of neurons connections we will focus on. The architecture of the neuron connections is the topology of the NN.

In addition to neural connection types, a NN is also characterized by its three types of layers: the input layer, the hidden layers, and the output layer. The hidden layers are the layers in-between the input and output layers. If there are many hidden layers, where more than three is considered many, then the NN is considered a deep neural network (DNN).

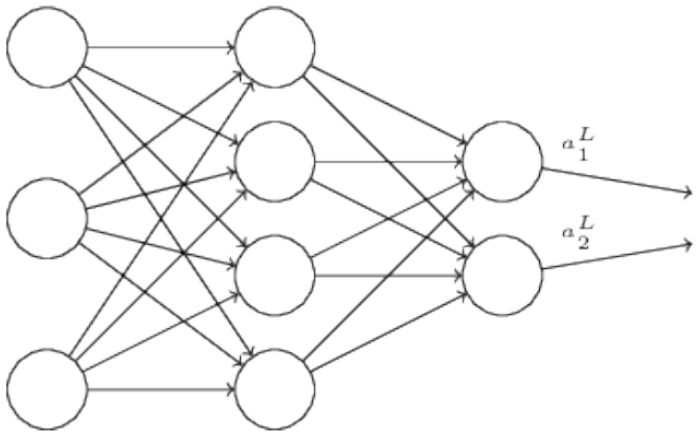

Figure 4.1: Here is an example of a shallow (not that many hidden layers) and fully-connected layers (each neuron in the previous layer connects to each neuron in the next layer) NN with only one hidden layer, Chapter 2 of [7].

These are the types of NNs that find practical applications. "Shallow" NNs are only useful for specific problems. ML methods perform better than shallow NNs for more practical problems. As such, DNNs are the main focus of this and many works.

NNs are used to answer a question in the form of input to output. The problem is finding a mapping from $R^n \rightarrow R^m$ where n is the dimension of the input vector and m is the dimension of the output vector. However, NNs need to "learn" in order to answer this type of question. To learn, NNs need input-output data pairs as well as a learning method.

The NN is subjected to a learning or training stage. Afterwards, the NN is usually tested with another set of data, usually separate from the one used to train it, for accuracy. It is important to note that NNs are only good for interpolation and are not meant to be used for extrapolating data. For example, NNs can be used for image recognition by using the image pixels as input. A NN can be taught to learn to identify an image as a cat in a binary manner, such as yes or no as its output. As such, this NN would not be able to say, for example, whether an image of animal was a duck or not. The hyper-parameters of NNs are the number of layers and the number of neurons per layer. Along with the NN topology,

the hyper-parameters do not change during the learning stage of the NN. The parameters of the NN are the weights and the biases. These parameters are adjusted by the learning method during the learning stage. Each neuron produces an output, for it needs to pass information to another neuron, that can be concisely put as follows (Chapter 2 of [7])

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \tag{4.1}$$

Here, the output a_j^l is that of the j^{th} neuron in the l^{th} layer. As such, a_k^{l-1} is the output from the k^{th} neuron of the previous layer. The values w_{jk}^l are the weights. These can be thought of as the connections from the previous layer to the next layer, connecting neuron k from Layer $l-1$ to neuron j of Layer l. As the summation suggests, there is one weight for each input a_k^{l-1}. Next, the bias b_j^l, a number greater than zero, is added to the total summation and corresponds to the j^{th} neuron in the l^{th} layer. Finally, σ is the so-called activation function. The previous summation is the input of this function. These activation functions are meant to be non-linear, for its input is already so. The role of activation functions is to allow for non-linearity in the NN. NNs are meant to approximate an unknown function from $R^n \to R^m$ that fits the given data. The likely non-linearity of this unknown function is dealt with the non-linearity of the activation functions. Some common activation functions are shown below.

Sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4.2}$$

Hyperbolic tangent:

$$\sigma(x) = \frac{2}{1+e^{-2x}} - 1 \tag{4.3}$$

Rectified linear unit:

$$\sigma(x) = \max(0, x) \tag{4.4}$$

The learning method aims to minimize the difference between the outputs of the network and the outputs of the training data. Since the NN is essentially a network composition of

functions, its gradient can be used to minimize the error of its output. Some common cost functions for the error are the mean square error (MSE) and the root mean square (RMS). One common algorithm to minimize the error is the so-called gradient descent through back-propagation. In this algorithm, staring from the last layer and ending at the first layer, the weights and biases are updated by following the negative gradient of the cost function with respect to the weights and bias of the current layer being updated.

4.2 Theory Behind Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of NN with an architecture better suited for image recognition. These types of NNs are very distinct from the one shown in Figure 4.1, for they are not fully-connected like.

For context, images are made up of pixels arranged in a grid. The resolution of an image is the amount of pixels it is composed of, for more pixels leads to greater detail in the image but also more memory space in storage. The output value of a pixel can be a shade of a color or just simply a color in the grayscale. The latter can take up less memory in storage if stored in a file format that takes advantage of the monochromaticity. As such, we will concern ourselves only with monochromatic images. Keeping the arrangement of pixels in a grid and, for our purposes, the value they store in mind, we can look at the architecture of CNNs and why CNNs are suited for image recognition. As explained in Chapter 6 of [7], CNNs use three basic ideas for this: local receptive fields, shared weights and biases, and pooling.

4.2.1 Local Receptive Fields

The inputs of a CNN are the pixels of an image. Local receptive fields are small groups of pixels located in the same general region. These local receptive fields take the output of a $p \times q$ region of pixels and convert it into one value for one neuron in a so-called convolutional layer to take as input. The purpose of this is to identify a feature (i.e., a characteristic of

the image such as a cat's ear for identifying images of cats, although this region of pixels would be too big for use in practice) of the image. Because the region of pixels is greater than 1, the data is reduced and the number of neurons in one convolutional layer is less than the number of pixels. In addition, because an image, in general, has several features, it is necessary to have several convolutional layers that are each independently connected to the input layer of pixels but not to each other, making most CNNs deep NNs. In this way, different convolutional layers, also called feature maps, identify different features.

4.2.2 Shared Weights and Biases

Because a feature can be located in different regions of the image, all the neurons in a convolutional layer share the same weights and bias. Because of this, the shared parameters are often collectively called filters or kernels. Filters ensures a degree of translational invariance of the images because each neuron in the same convolutional layer detects the same feature, just at a different position in the image. The output of a neuron in a convolutional layer with shared weights and bias has the form

$$a_j^l = \sigma\left(b + \sum_{l=0}^{p}\sum_{m=0}^{q} w_{l,k} a_{j+l,k+m}\right) \tag{4.5}$$

where p and q are the dimensions of the rectangular region of pixels taken as input for a neuron. The variables have the same meaning as those in Equation (4.1).

4.2.3 Pooling

Lastly, pooling is done in the so-called pooling layers. These layers are used right after the convolutional layers, and their role is to "condense" the output of convolutional layers. This is done by taking another small region of the neurons in the convolutional layers and taking some aggregate value from the group of neurons. Two operations for these are taking the maximum neuron output or taking the square root of the sum of the squares of the neurons' outputs, each known respectively as max-pooling and L2 pooling. As with

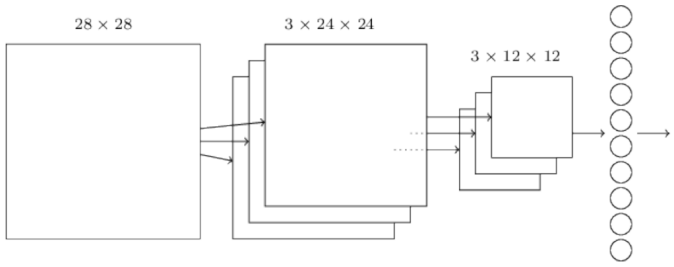

Figure 4.2: Example diagram of a CNN with an input layer, convolutional layers, pooling layers, and an output layer, Chapter 6 of [7].

the convolutional layers with the input pixels, the pooling layers are smaller than the convolutional layers in terms of number of neurons. Pooling layers can be thought of as affirming that a feature is indeed located in a certain region of the pixels but afterwards disregarding the positional information. This is so that the "condensed" information can be fed to the output layer more easily (less parameters needed and thus easier training of the network). Each convolutional layer is connected to a pooling layer, there are no connections between pooling layers, and pooling layers can be fully-connected to the output layer. An example CNN topology shown in Figure 4.2.

4.3 Enclosures and Search Algorithms

In this section, we go over the methods and algorithms that will be used to aid the CNN described in the previous section. The necessity of these algorithms will be explained in chapter 5, the implementation of the proposed procedure of this work.

4.3.1 Enclosures

As described in Chapter 4 of [29], for a system of equations of the form

$$A^I x = b^I \tag{4.6}$$

an enclosure is an interval vector $[\underline{y}, \overline{y}]$ satisfying

$$X \subseteq [\underline{y}, \overline{y}] \tag{4.7}$$

where X is the set

$$X = \{x; Ax = b \text{ for some } A \in A^I, b \in b^I\} \tag{4.8}$$

For our purposes, we will consider only the general idea of enclosures containing the true value of any general problem, not just unresolved systems of linear equations. In addition, we will consider scalars instead of vectors, although the idea of vectors can be extended to include scalars as one-dimensional vectors. In particular, we note that having a smaller enclosure leads to greater accuracy in the true value of our solution if we consider the midpoint of the enclosure as our solution y, where, for scalar solutions, the midpoint would be

$$y = (\underline{y} + \overline{y})/2 \tag{4.9}$$

This leads the absolute error from the true value $\tilde{y}$ to be

$$|\tilde{y} - y| \leq (\overline{y} - \underline{y})/2 \tag{4.10}$$

where we reiterate that $\underline{y}$ is the lower bound of the enclosure and $\overline{y}$ the upper bound.

4.3.2 Search Algorithms

As described in Chapter 3 of [8], a search algorithm aims to find a key k within a data structure D of records. Many times, D is a so-called *associative array* (also known as a *table* or *dictionary*). Associative arrays map a disjoint set of keys to an arbitrary set of values. A table can have order or not, and it is defined by a set of ordered pairs (k, v) where k is the key and v is the value associated with it. In particular, a table must not have identical keys since entries are identified by their key.

Two types of search algorithms to consider are the sequential search algorithm and the binary search algorithm. Both search algorithms work best if the table is ordered (i.e., the keys, depending on their type, are ordered). For our purposes, we can assume this.

The sequential search algorithm just starts at the of the head of the table and examines each element until it find the correct key. The matching of the key can be determined by its associated value as well. For the purposes of determining the melting line of metals, this will be the case.

In contrast, the binary search algorithm looks first at the middle element of the table and determines if this element is the desired one, i.e. $k = m$ where m is the desired value. If this is the case, the binary search terminates, otherwise if $k < m$ then a recursive call is made on the head sublist, and if $k > m$ then the recursive call is made on the tail sublist instead. This process is repeated until $k = m$. For our purposes, there will be a tolerance such that the upper bound of Equation (4.10) is given. So instead, the error of k from m must be smaller than a given tolerance. An example of the binary search algorithm is given in Figure 4.3.

It is easy to prove that given the length n of a table, the sequential search algorithm takes $O(n)$ time-complexity for a successful or unsuccessful find while the binary search algorithm takes $O(\log n)$ time complexity. We will concern ourselves only with the binary search algorithm because it is faster yet still relatively simple.

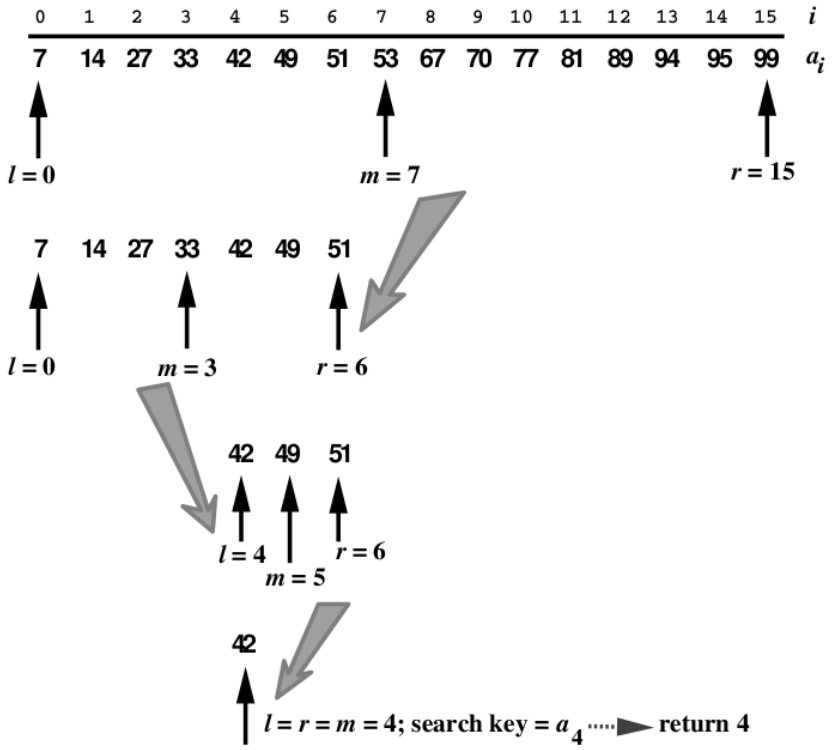

Figure 4.3: Example of the binary search algorithm where the key $k = m = 42$ is sought, Chapter 3 of [8].

Chapter 5

Proposed Method for Defining Transition Lines

As introduced in the introduction of this paper, we seek to determine the melting line of metals in a more efficient and automated way. To this end, we will use enclosures, binary search, and CNNs. The use of CNNs has been done before in [30], however, in this paper, they used unsupervised CNNs for classification. In our case, our CNNs will use training data to categorize. For this end, we will explain how the methods of Section 4.3 will be used to make the CNN more effective. After, we explain how our CNN will be used as well as present its architecture. Lastly, we will present how we put all these methods together with the computer languages used for their implementation in actual code and a pseudo-code.

5.1 The Proposed Search Algorithm and Enclosure Method to Reduce Computation Time

As mentioned in Subsections 4.3.1 and 4.3.2, we will make use of enclosures and the binary search algorithm to speed up the determination of the melting line as compared to the procedure described in Section 3.4.

5.1.1 Binary Search Implementation for the Finding the Melting Temperature

As mentioned in Section 3.4, the temperature ranges were divided into rather numerous intervals. For a given lattice constant, and an initial interval from 3200 to 7200, 29 simulations were needed to calculate the melting temperature within one Kelvin. In this sub-section, we want to prove that using the binary search is optimal because we minimize the number of simulations we need to run to find the melting temperature within a given tolerance/error. Given an initial temperature interval $[T_L, T_U]$ that we seek to repeatedly divide into smaller sub-intervals until the size of the sub-intervals is of length e, were we define the length ΔT of an interval as in Equation (5.1),

$$\Delta T = T_U - T_L \tag{5.1}$$

then we can prove that the number of iterative divisions d that leads to the least number of sub-intervals is 2. That is, given the task of dividing an interval of length ΔT until it is of size e or smaller, then the number of sub-intervals we must produce for each division is 2. First we note that the number of divisions has to be a natural number since we do not define decimal divisions in any way. Now we show that in order for the next sub-intervals to be smaller than the previous sub-intervals, the previous sub-intervals have to be divided into at least two sub-intervals as shown in Equation (5.2)

$$d \geq 2 \tag{5.2}$$

We also note that all the sub-intervals from the same division are of equal length. Thus, we can conclude that the least number of divisions of a sub-interval such that the resulting sub-intervals are smaller is 2 because the infimum of the set $\{d \geq 2 \mid d \in N\}$ is 2.

This implies naturally that the binary search algorithm of Subsection 4.3.2 should be used, where instead of a numerical comparison indicating which sub-interval should be

searched, the melting in the graph of the temperature versus time graph mentioned in section 3.4 should be used instead. If the graph shows no melting, then the head sub-interval should be searched, and if the graph shows melting, then the tail sub-interval should be searched instead.

Two is the optimal number of iterative divisions for the determination of the melting line procedure because, as mentioned in Section 3.4, a simulation set at a certain temperature takes several minutes to complete, and the total time is shown in Equation (5.3)

$$\tau_{tot} = N_S \tau_{avg} N_{latt} \tag{5.3}$$

where N_S is the total number of simulations needed so that the melting temperature is within the desired temperature range length Δr, N_{latt} is the number of lattice constants lengths, and τ_{avg} is the average time of all the simulations. From practice, all simulations with the same number of steps take about the same amount of time, regardless of temperature or lattice constant. If we minimize d, we minimize N_S, and thus we minimize τ_{tot}. τ_{avg} is fixed because it depends on the number of processors used and the number of steps for time-stepping. N_{latt} is also fixed because we desire a certain number of data points for a good interpolation of the Simon-Glatzel equation. As such, in Equation (5.3), only N_S can be minimized. Given that $d = 2$ for optimal search, we can see that given a desired interval length of err (melting temperature error), the number of division N_d on an initial interval length $\Delta T_{initial}$ to reach e can be determined by Equation (5.4)

$$N_d = \log_2(\Delta T_{initial}) - \log_2(err) \tag{5.4}$$

It is important now to more specifically explain how this binary search will be implemented. In particular, we make use of Equations (4.9) and (4.10) from Subsection 4.3.1 regarding enclosures to further optimize the binary search. The temperature interval $[T_L, T_U]$ is assumed to be an enclosure such that it is ensured that melting will not occur at a temperature of T_L and melting will occur at a temperature of T_U. This can be achieved by taking a $[T_L, T_U]$ to

be very large and using some experimental information, such as the melting temperature of the metal at 0 GPa, i.e. at the natural lattice constant of the crystal, to determine what is considered large. First, we run a simulation for a system at a temperature of T_L. Afterwards, we run another simulation using the phase of the previous simulation, as explained in Section 3.4, with temperature T_m, where T_m is the midpoint of T_L and T_U. Thus, have to consider three values, T_L, T_m and T_U in the binary search. Recalling the aforementioned rule for determining which sub-interval, either the head or tail sub-interval, to search next, T_m must replace either T_L or T_U for the next sub-interval. T_L if melting does not occur at T_m and T_U if it does. This process is repeated until the temperature interval length ΔT_k at the k^{th} division is less than or equal to $2e$. To make use of enclosures and Equation (4.9), which places our temperature at the best place to have a max boundary on the error from the true melting temperature, Equation (4.10), we run one more simulation at the midpoint of the interval having length $2e$ to fulfill the maximum error of e. As a result, we see that the relation between N_d and N_S for this procedure is

$$N_S = 1 + N_d \tag{5.5}$$

5.1.2 Updating Upper Boundary of Next Longest Lattice Constant: Shrinking Enclosure

Another optimization can be made regarding the initial temperature range length $l_{initial}$ of each lattice constant. The boundaries $[r_L, r_U]$ corresponding to $l_{initial}$ are the temperature range where the melting temperature is searched. We can make $l_{initial}$ lattice-constant dependent and add the a subscript to it as shown here: $l_{initial,a}$. If we make the assumption that at a higher pressure the melting temperature is higher, then we can start the binary search at the shortest lattice constant and find its melting temperature first. After this, we can use this melting temperature as the upper boundary of the next lowest lattice constant since its melting temperature has to be higher than the next lowest lattice constant. Using

this, we can repeat this procedure for the subsequent melting temperatures searches as shown in Equation (5.6)

$$r_{upper,a_p-\Delta a} = T_{melting,a_p} \tag{5.6}$$

Here, a_p is a particular lattice constant in the list of lattice constants. We can assume that the set $\{a_p\}$ is ordered from smallest to largest and the difference between lattice constants in this set is a constant value Δa. Thus, we can continually shrink the $l_{initial,a}$ of the bigger length lattice constants. This will help reduce the number of $N_{d,a}$ for each lattice constant a and thus speed the process at least some more.

5.2 Using Convolutional Neural Networks to Partially Automate Procedure

5.2.1 Reason for Using Convolutional Neural Networks as Opposed to Normal Neural Networks

The CNNs mentioned in Chapter 4 will be used for the classifying of melting or not melting of a system, and thus provide the evaluation of the binary variable that will be used for the binary search algorithm of Subsection 5.1.1. The data form the procedure in Section 3.4 will be used to train the CNN. This will be like a human trying to identify graphs like those in figure 5.1 for melting and graphs like those in figure 5.2 for no melting. We note that the axis labels and interval numbering in the graphs are removed. This greatly simplifies the learning of the CNN.

As mentioned in Subsection 4.2, we will stick with gray-scale images because they take up less memory space. In addition, the images will be jpg files and have 72 dots per inch (dpi) for the resolution and be 480×480 pixels in size as mentioned in [31]. This enables us to set the size of the file regardless of the size of the original data file. This means that for sufficiently big data files, the graph images will take less space and thus be superior for

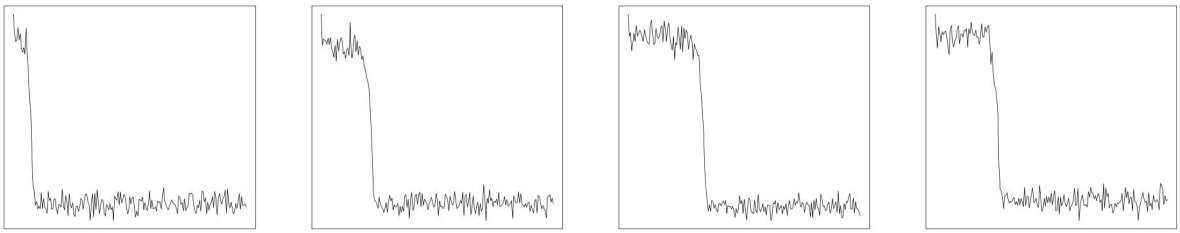

Figure 5.1: Sample of training data for the CNN to recognize melting.

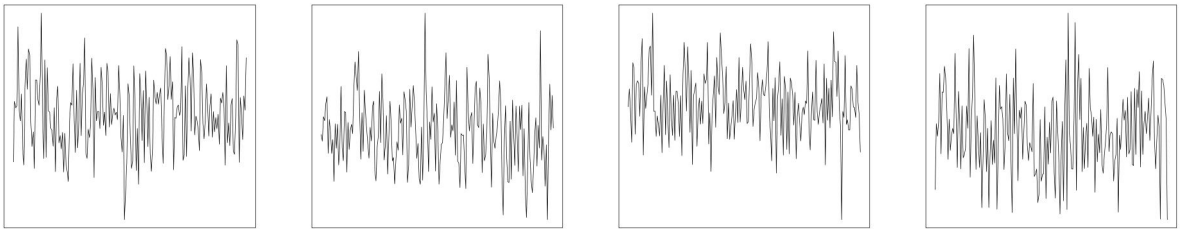

Figure 5.2: Sample of training data for the CNN to recognize no melting.

storage. The data file size is determined by the frequency a data point is recorded and by the number of steps in the simulation.

The previous paragraph gives one argument in favor of not building a non-convolutional NNs and just feed the data files directly into it. If the simulations need to be longer to capture the melting, then the image jpg files have the potential to take up less memory space. Another argument for using a CNN as opposed to a conventional NN is that NNs (including CNNs) are not perfect and can mislabel a graph. Most NNs are rated by a percent accuracy when identifying the correct value for the training data. The only way for a human to check for potential errors in our melting line determination procedure is to look at the temperature versus time graphs. This allows a user to more easily understand why a CNN would label a graph a certain way. This is noted in Chapter 7 regarding partial melting.

However, a data transformation will be applied to the data files before plotting: the first 1% of the data points will be deleted. This is done because, when doing the velocity re-scaling for the system's target temperature, the MD takes time to equipartition the total energy into the kinetic and potential energies. Because the temperature is determined only by the kinetic energy, the temperature at first is as high as that suggested by Equation (2.3). This produces at least one outlier, so, when the data is plotted, the temperature axis is scaled to include this or these outliers (depending on the time-step and the time it takes for the system to equilibrate to the wanted temperature, there could be more than one outlier). As such, the visual representation of the rest of the data is scaled down in size, and thus the "melting feature" ("feature" as described in Subsection 4.2.1, Local Receptive Fields) is more difficult to "see" (by both humans and the CNN). Removing the outlier(s) solves this issue.

5.2.2 Building our Convolutionl Neural Network in Python

There are two popular machine learning packages for building, for example, NNs in Python: PyTorch and TensorFlow. As mentioned in the blog post [32], TensorFlow is easier to implement, and because our CNN has to categorize graphs into just two types: melting and non-melting, we can go with the easier but less dynamic machine learning implementation of TensorFlow.

The online guides used for building our CNN come from [33], [34], and [35]. Our CNN is composed of two convolutional layers of 64 neurons each and ReLU activation functions and strides of 3 by 3. Each of these are followed by a max-pooling layer of pool-size 2 by 2. These two parallel convolutional-pooling layer pairs are both connected to a regular densely-connected layer (like that of Figure 4.1) of 64 neurons with regularizer activation function. Finally, this layer is connected to another regular densely-connected of 1 neuron for the output layer of 0 or 1 with sigmoid activation function. The optimizer is the so-called Adam algorithm and the loss function is the binary cross-entropy.

The CNN will be trained with 55 melting graph images and 126 not-melting graph

images, for a total of 181 training data points. The images are originally of 480 by 480 pixels, but these are reduced to images of 300 by 300 pixels to be fed to the CNN. The CNN was trained with 4 epoch and with batch sizes of 32 data points.

5.3 Implementation of Proposed Procedure

Our implementation will be a wrapper program that makes use of python, R language, shell bash, and LAMMPS scripts. The pseudo-code is provided in Algorithms 1, 2, and 3. The code can be downloaded from github at https://github.com/caibarra5/LAMMPS-Wrapper-Machine-Learning-Assisted-Melting-Line-Determination-for-Metals.

Algorithm 1 Calculate Melting Line Points of Metal part 1

Get $steps\_per\_run$
Get $steps\_per\_data\_point$
Get $steps\_per\_dump$
Get $initial\_range[\ ]$ array
Get $user\_input\_error\_tol$
for $file$ in List of Needed Files **do**
  if $file$ does not exist **then**
    Terminate program and throw error.
  end if
end for
$error\_tol = 2 * user\_input\_error\_tol$
Put list of lattice constants into array $latts$
$ca\_array\_len = length(latts)$
$ca\_last\_index = ca\_array\_len - 1$
Set $output\_file$
for $i = 0; i < \$ca\_array\_len; i++$ **do**
Generate lattice directory for $latts[i]$
Put initial temperature range into array $initial\_range$
$l\_bound = initial\_range[0]$
$u\_bound = initial\_range[1]$
$m\_point = (l\_bound + u\_bound)/2$
Run LAMMPS script $melt\_intial\_run$ given
  $prev\_temp = null$
  $temp = l\_bound$
  $a = latts[i]$

Algorithm 2 Calculate Melting Line Points of Metal part 2

$n = steps\_per\_run$
$k = steps\_per\_data\_point$
$l = steps\_per\_dump$
$dt = dt$
$restart\_file = null$
Remove first 1% of data points in dumped thermo. data file
Graph data without labels or intervals with R script
Categorize melting with CNN in python script
$melted\_signal = output\ from\ CNN$
print $melted\_signal$
if $melted\_signal = 1$ **then**
Exit program and output error: Lower bound too high, need to lower it.
end if
$error = u\_bound - l\_bound$
if $error < error\_tol$ **then**
$error\_less\_than\_tol = 1$
else
$error\_less\_than\_tol = 0$
end if
while $error\_less\_than\_tol = 0$ **do**
Run LAMMPS script $melt\_iterative\_run$ given
$prev\_temp = l\_bound$
$temp = m\_point$
$a = latts[i]$
$n = steps\_per\_run$
$k = steps\_per\_data\_point$
$l = steps\_per\_dump$
$dt = dt$
$restart\_file = l\_bound\_restart\_file$
Remove first 1% of data points in dumped thermo. data file
Graph data without labels or intervals with R script
Categorize melting with CNN in python script
Move thermo. data file and corresponding graph file into lattice directory for $latts[i]$
$melted\_signal = output\ from\ CNN$
print $melted\_signal$

Algorithm 3 Calculate Melting Line Points of Metal part 3

if $melted\_signal = 0$ **then**
remove restart file from l_bound
$l\_bound = m\_point$
else if $melted\_signal = 1$ **then**
remove restart file from u_bound
$u\_bound = m\_point$
end if
$m\_point = (l\_bound + u\_bound)/2$
print l_bound
print m_point
print u_bound
$error = u\_bound - l\_bound$
if $error < error\_tol$ **then**
$error\_less\_than\_tol = 1$
else
$error\_less\_than\_tol = 0$
end if
end while
$m\_point = (l\_bound + u\_bound)/2$
Run LAMMPS script $melt\_iterative\_run$ given
$prev\_temp = l\_bound$
$temp = m\_point$
$a = latts[i]$
$n = steps\_per\_run$
$k = steps\_per\_data\_point$
$l = steps\_per\_dump$
$dt = dt$
$restart\_file = l\_bound\_restart\_file$
Remove first 1% of data points in dumped thermo. data file
$melt\_temp = average(last\ 100\ temperature\ data\ points\ from\ thermo.\ file)$
$melt\_press = average(last\ 100\ pressure\ data\ points\ from\ thermo.\ file)$
Store latts[i], $melt\_press$ and $melt\_temp$ in output file
Move thermo. data file into $latts[i]$ directory
Set $initial\_range[1] = m\_point$
end for
Print wall-clock time

Chapter 6

Results of New Procedure on Copper and Other Metals

6.1 Copper

We run our code using the same list of lattice constants as that of Table 3.1 and use the algorithm described in Subsection 5.3 to compute the melting pressure and temperature. From several runs of our procedure, we conclude that without user intervention in the number of steps, the algorithm cannot be as accurate as the manual computation. The error tolerance that seems to work best is around 100K. Any smaller error makes it trickier to determine the number of steps necessary to see the melting feature. As such, the CNN does not see melting and thus keeps outputting a 0 for melting, leading to the program to search for higher melting temperatures until it reaches a distance of the error tolerance from the upper bound. The results can be seen in Table 6.1.

Table 6.1: Melting temperature and pressure results of Cu from proposed procedure for an error in melting temperature of 100K.

Latt. Const. (A)	Melt. Press. (GPa)	Melt. Press. Err. (GPa)	Melt. Temp. (K)	Melt. Temp. Err. (K)
3.150	219.298	0.23098	5825.62	58.5191
3.185	190.276	0.18646	5448.58	50.7609
3.220	165.175	0.21501	5218.57	50.5499
3.255	141.019	0.16025	4707.11	37.0924
3.290	121.337	0.19307	4396.74	45.1702
3.360	88.523	0.11294	3803.45	29.3288
3.395	75.317	0.13528	3534.96	32.4642
3.430	63.200	0.09132	3249.70	23.4639
3.500	43.435	0.09828	2795.18	22.4904
3.600	22.344	0.08188	2150.48	16.1339

Furthermore, the wall-clock time of this run was just 26 minutes and 28 seconds as compared to the 66 hours of the manual computation. The proposed procedure is 150 times faster but the accuracy is much greater in the manual determination (100 times more accurate at least). The reason for this speed up is that the number of steps that worked best for each simulation in the proposed method, regardless of change in temperature ΔT from the previous run, was about 10000 steps. This is much less than the 10^5 to $3 \cdot 10^5$ range used for the manual determination of the melting line. In addition, it seemed that a large ΔT from the previous to the current run helped the melting feature be present. This could be because, along with the 10 ps corresponding to the 10000 steps, the system does not have time to equilibrate to the temperature of the previous phase. That is, the "springs" between atoms could more easily be broken if the kinetic energy available allowed it to be so without having to wait for the system to equilibrate. A shorter simulation time may have helped this melting feature in the graph from being scaled down.

The best comparison that can be between the normal and proposed methods is the plot of the Simon-Glatzel equations as shown in Figure 6.1. The bars for each data point correspond to the standard deviation of the data points used to calculate the average temperature corresponding to the melting temperature. These bars were also included for the graphs in Figures 6.3 and 6.5 for aluminum and tantalum respectively. The two methods for copper are fairly close.

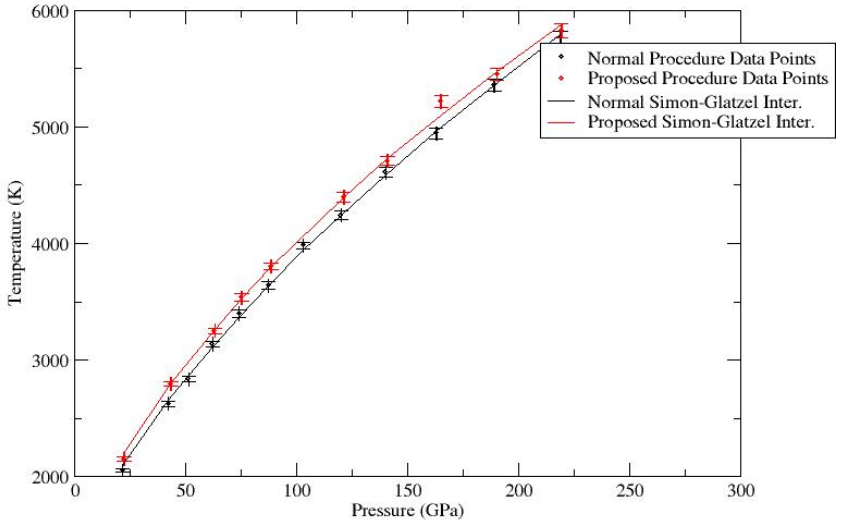

Figure 6.1: Normal method compared with proposed method for Copper melt-line determination data points along with the interpolated Simon-Glatzel curve.

Table 6.2: Results of proposed procedure for melting line of Cu for an error in melting temperature of 100K.

Latt. Const. (A)	Lower Bound (K)	Midpoint (K)	Upper Bound (K)	Interval Size (K)	Melting Signal
3.150	6625.0000	7187.5000	7750.0000	1125.0000	0
3.185	6531.6161	6583.8012	6635.9862	104.3701	1
3.220	6234.8136	6322.0605	6409.3074	174.4938	1
3.255	5656.8029	5739.9601	5823.1173	166.3144	1
3.290	5295.5889	5369.6507	5443.7126	148.1237	1
3.360	4526.8713	4587.6794	4648.4875	121.6162	1
3.395	4139.2195	4363.4495	4587.6794	448.4599	0
3.430	3869.4428	3979.8059	4090.1691	220.7263	1
3.500	3319.1357	3401.9620	3484.7883	165.6526	1
3.600	2501.2263	2651.3489	2801.4715	300.2452	1

In Table 6.2 we see the temperature range used for the melting temperatures of Figure 6.1 along with the melting signal. A melting signal of 0 means the system did not melt according to the CNN while a 1 means it did. This is why this work is titled "assisted" instead of "automated" mapping of the melting line. All the thermodynamic data files

used for Figure 6.1 and Tables 6.1 and 6.2 showed melting of the system, so all the melting signals in Table 6.2 should be 1 but the CNN has an about 80% accuracy with this system. The CNN cannot be perfect, and its best use is by allowing un-supervised determination of what temperatures need to be used for subsequent simulations in the binary search algorithm. After each full run of the proposed procedure, a series of data files and images of graphs are kept, and the user can see all these graphs and determine which data values to use for post-processing.

Table 6.3: Results of proposed procedure for melting line of Cu for an error in melting temperature of 100K.

Midpoint (K)	Melting Signal
1000.0000	0
3184.8254	0
4277.2380	0
4823.4443	0
5096.5475	1
4959.9959	1
4891.7201	0

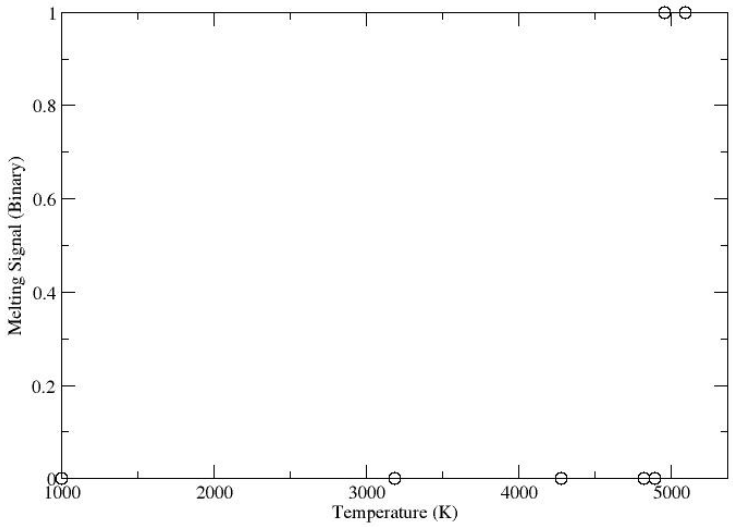

Figure 6.2: The melting signal (open circles) is plotted against the temperature, where a value of 0 indicates the system melts at that temperature, and a melting signal of 1 indicates it does not.

Figure 6.2 shows an example of the binary search of Copper. Here, the initial range is from 1000 to about 5369.6508. Table 6.3 shows the exact values and the sequential order of the search.

6.2 Aluminum

Here we use the potential as described by [36]. The melting temperature of aluminum at 0 GPa is 933.5K. This point in the T-P graphs is used for the Simon-Glatzel equation interpolation. The user error tolerance was 100K and the wall-clock time was 1 hour, 1 minute and 47 seconds. The data produced is in Table 6.4, plotted in Figure 6.3, and information about the melting temperature to evaluate the CNN is provided in Table 6.5.

Table 6.4: Results of proposed procedure for melting line of Al for an error in melting temperature of 100K.

Latt. Const. (A)	Melt. Press. (GPa)	Melt. Press. Err. (GPa)	Melt. Temp. (K)	Melt. Temp. Err. (K)
3.50	109.1700	0.0612375	3424.48	25.6206
3.55	92.0788	0.0424759	3201.69	23.6320
3.60	77.2372	0.0555393	2996.82	27.2949
3.65	64.3363	0.0447707	2764.82	18.3809
3.70	53.5704	0.0523602	2644.90	20.0571
3.75	43.9180	0.0464520	2441.63	19.2482
3.80	36.4674	0.0570211	2428.61	23.4342
3.85	28.3400	0.0529603	2080.38	20.0343
3.90	22.5703	0.0438710	1958.89	14.5535
3.95	16.9506	0.0422857	1721.85	13.2672
4.00	12.6823	0.0401613	1606.57	15.0156

Figure 6.3 can be compared with Figure 6.4, which uses shock compression under static conditions [9]. However, we can still see that the two curves agree fairly well.

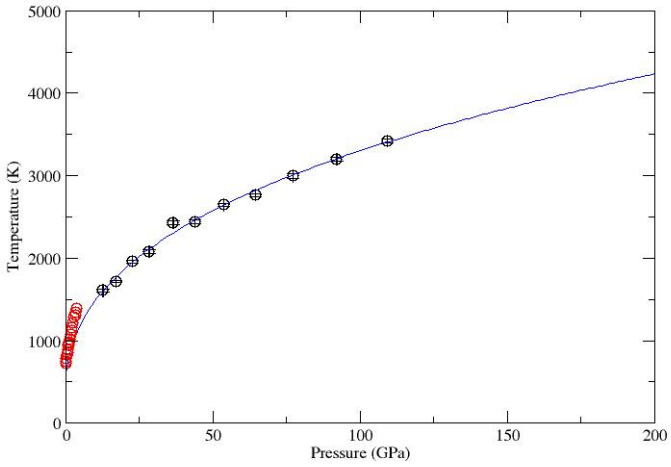

Figure 6.3: Proposed method melt-line determination of Aluminum data points (black circles) with Simon-Glatzel interpolation curve (black curve) and melting points from more accurate phase-coexistence method [4] (red circles).

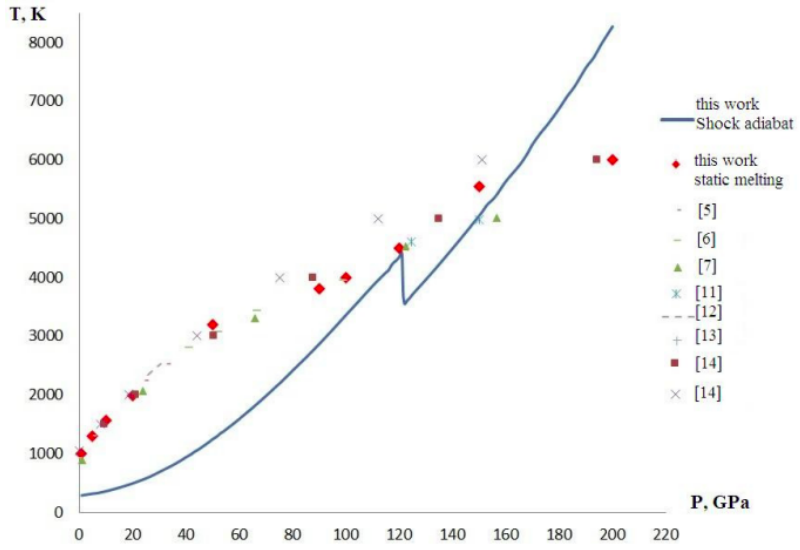

Figure 6.4: Aluminum melt-lines and Hugoniot lines with more accurate methods than the z-method [9].

Table 6.5: Melting information for binary search for final melting temperature of Aluminum at given lattice constant.

Latt. Const. (A)	Lower Bound (K)	Midpoint (K)	Upper Bound (K)	Interval Size (K)	Melting Signal
3.50	4000.0000	4250.0000	4500.0000	500.0000	0
3.55	3578.1250	4007.8125	4437.5000	859.3750	1
3.60	3584.8389	3677.1546	3769.4703	184.6314	1
3.65	3342.5102	3426.1713	3509.8324	167.3222	1
3.70	3122.8999	3274.5356	3426.1713	303.2714	1
3.75	2923.8780	2992.5879	3061.2978	137.4198	1
3.80	2743.5144	2868.0512	2992.5879	249.0735	0
3.85	2447.7397	2568.3846	2689.0296	241.2899	0
3.90	2323.3245	2374.2216	2425.1187	101.7942	1
3.95	2030.6662	2116.5550	2202.4439	171.7777	1
4.00	1837.4162	1976.9856	2116.5550	279.1388	1

6.3 Tantalum

Here we used the potential as described by [10]. The melting temperature of tantalum at 0 GPa pressure is 3293K. This point in the T-P graphs is used for the Simon-Glatzel equation interpolation. The user error tolerance was 100K and the wall-clock time was 29 minutes

and 49 seconds. The data produced is in Table 6.6, plotted in Figure 6.5, and information about the melting temperature to evaluate the CNN is provided in Table 6.7.

Table 6.6: Results of proposed procedure for melting line of Ta for an error in melting temperature of 100 K.

Latt. Const. (A)	Melt. Press. (GPa)	Melt. Press. Err. (GPa)	Melt. Temp. (K)	Melt. Temp. Err. (K)
3.000	117.412	0.13423	4136.61	50.1941
3.025	103.889	0.10475	4255.47	49.0543
3.050	90.674	0.11757	4196.96	49.5546
3.100	66.424	0.10153	3928.98	46.0076
3.125	55.853	0.10042	3831.81	42.6721
3.150	46.114	0.10408	3661.64	42.4867
3.175	38.031	0.09060	3758.83	40.0053
3.200	30.142	0.08365	3521.98	41.9185
3.225	23.503	0.09059	3427.50	43.6133
3.250	17.977	0.10810	3443.49	43.0064

Figure 6.5 can be compared with Figure 6.6, which uses more accurate methods to the z-method, including a DFT method, for determining the melting line of tantalum.

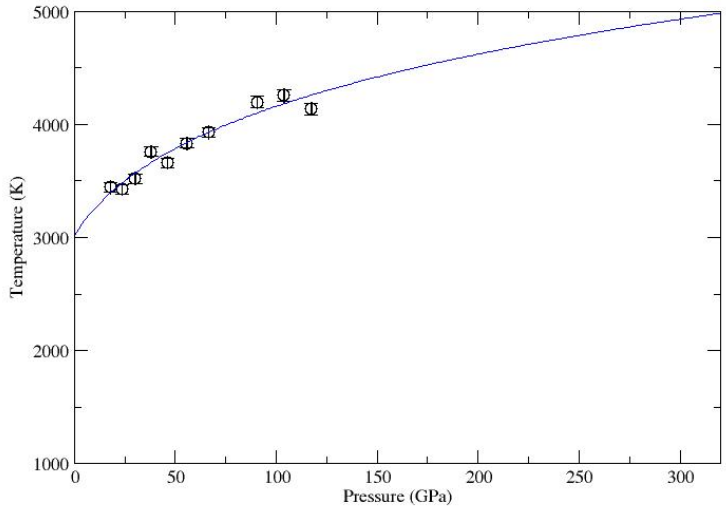

Figure 6.5: Simulation data (black circles) with errors and Simon-Glatzel interpolation (blue curve) for Tantalum. Red circles make use of two-phase coexistence [10].

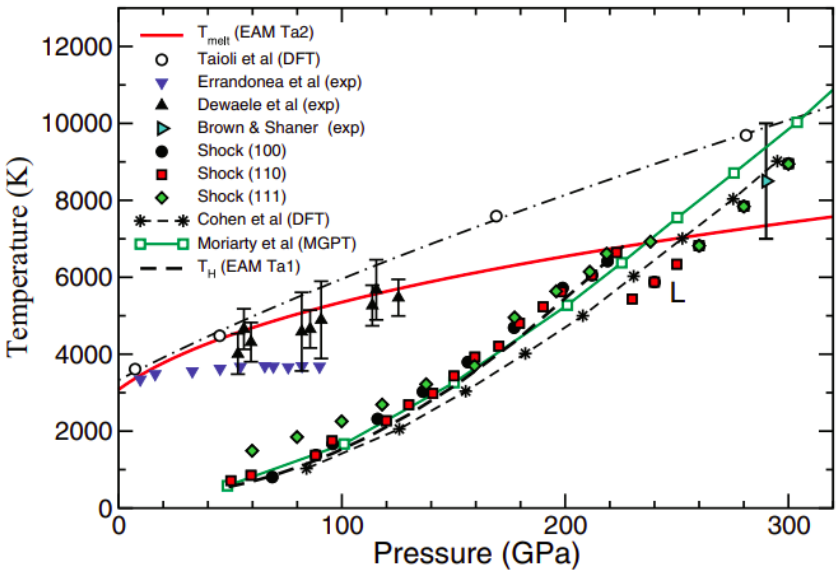

Figure 6.6: Tantalum melt-lines and Hugoniot lines [10].

As can be seen, both graphs agree fairly well.

Table 6.7: Melting information for binary search for final melting temperature of Tantalum at given lattice constant.

Latt. Const. (A)	Lower Bound (K)	Midpoint (K)	Upper Bound (K)	Interval Size (K)	Melting Signal
3.000	5125.0000	5296.8750	5468.7500	343.7500	0
3.025	5108.8867	5177.3682	5245.8496	136.9629	1
3.050	4916.2827	5046.8255	5177.3682	261.0855	0
3.100	4674.3834	4796.8628	4919.3423	244.9589	0
3.125	4616.9711	4677.2540	4737.5368	120.5657	0
3.150	4447.4256	4504.8827	4562.3398	114.9142	1
3.175	4395.3551	4450.1189	4504.8827	109.5276	0
3.200	4234.4865	4342.3027	4450.1189	215.6324	0
3.225	4183.9476	4237.0134	4290.0792	106.1316	0
3.250	4135.8567	4186.4351	4237.0134	101.1567	0

Chapter 7

Conclusion

As the implementation of the proposed method shows in Tables 6.2, 6.5, and 6.7, the CNN makes mistakes. The thermodynamic information data files were chosen manually such that full melting occurred. As such, all the melting signals should be 1s, but in all systems, there are a few 0s, especially in Tantalum. The benefit was that the data files were generated automatically, which is the optimal use of the proposed method and code implementation. The method is only meant to greatly aid the determination of the melting line of metals. The great difficulty of fully automating the procedure is explained below.

In reality, the CNN used in Algorithms 1 to **??** has a 80% accuracy with respect to the training data. However, for the copper re-run of Table 6.2, the accuracy is 80%. For aluminum, Table 6.5, it is 72%, and for tantalum it is 20%. If we take all systems into consideration, then there is a 39% accuracy. The lower accuracy for the new systems may be due to the CNN not being trained with their melting/non-melting data points. As such, the copper graphs don't generalize perfectly.

The tolerance used for all the data points found in Tables 6.2, 6.5, and 6.7 was 100K, meaning that the interval size for all these tables would ideally be 200K or less, but for many data points this is not the case, meaning the desired accuracy is not achieved and can only serve as a rough stop for the algorithm. Several parameters affect the accuracy of data points. The shape of the time versus temperature graph of a melting system is affected by the length of the simulation (the number of steps), the initial temperature range to search for melting temperatures, the user input tolerance, and even the list of lattice constants and the frequency of recording data points. Because the potential used for each system also determines the melting, the optimal values for all these parameters is not determinable

in practice. The best use of the proposed method and the code implementation is to have before-hand knowledge of the systems lattice constant and melting temperature at 0 pressure and determine a suitable range of lattice constants to find P-T data points. The melting temperature at 0 pressure can give a rough idea of the initial temperature range, but a run with the lowest and highest lattice constants alone to find better boundaries for the initial temperature range is essentially necessary. The rest of the parameters mentioned are set in the code, seem to work well and thus it is recommended they are not modified.

One would think that having a more accurate CNN would help, but it is better to train the CNN with new data and not use the same data. That is, run the program and use the new graphs generated to train the CNN. The motive for not training a CNN to have 100% prediction for the training data already used is to avoid over-training. Over-training the CNN leads to the CNN memorizing the "solutions" and thus impeding interpolation (new predictions for data points within the class of data the CNN was trained with). The determination of the optimal parameters and even the hyper-parameters of the CNN is a complex issue, and thus the CNN implementation in this paper was done in the simplest form, and the CNN parameter values were chosen so that the CNN worked fairly well. Only 6 batches (training cycles) were done and only an 80% accuracy was considered suitable to avoid over-training.

Chapter 8

Future Work

8.1 Replacement of CNN with Numerical Filter

The CNN comes with its downfalls when applied to the temperature versus time graphs of the systems. As such, we can propose another method for determining if a graph shows melting or not. This method makes use of normalizing the data by subtracting the minimum and dividing by the maximum. This method, a sort of filter, aims to apply the aforementioned data normalization at some percentage p of the first and last number data points. After, we compute the average of these two extremes of data point groups. Lastly, we subtract the average of the last normalized data points from that of the first normalized data points. Now, determine a value $0 < c < 1$. If the difference between averages is greater than c, then the system melted. If not, then it did not melt. We can call the set of all temperature data points $\{T_i\}$. As such we can say the following

$$T_{avg,-\%p} = avg(last\ \lceil \%p|\{T_i\}|\rceil\ of\ \{T_i\}) \tag{8.1}$$

$$T_{avg,\%p} = avg(first\ \lceil \%p|\{T_i\}|\rceil\ of\ \{T_i\}) \tag{8.2}$$

The data normalization can be done after computing the averages by simply applying the normalization to the averages.

$$T_{avg,-\%p,normal} = (T_{avg,-\%p} - \min(\{T_i\}))/(\max(\{T_i\}) - \min(\{T_i\})) \tag{8.3}$$

$$T_{avg,\%p,normal} = (T_{avg,\%p} - \min(\{T_i\}))/(\max(\{T_i\}) - \min(\{T_i\})) \tag{8.4}$$

Thus, $T_{avg,-\%p,normal}$ and $T_{avg,\%p,normal}$ must be between 0 and 1. In particular, $T_{avg,-\%p,normal} \approx 0$ and $T_{avg,\%p,normal} \approx 1$. Given these, we can see that if the system melts, the normalized drop in temperature,

$$\Delta T_{normal} = T_{avg,\%p,normal} - T_{avg,-\%p,normal} \tag{8.5}$$

would ideally be around 1. In reality, however, a number of situations can prevent this, such as the averages capturing not enough data points for $T_{avg,\%p,normal}$ to constitute a higher enough value than $T_{avg,-\%p,normal}$ or the system not melting completely. As such, it is ideal to have a value c as a threshold. Thus, the following condition can be used

Algorithm 4 Numerical Filter Replacement for CNN

```
if ΔT_normal > c then
    melted_signal = 1
else
    melted_signal = 0
end if
```

The disadvantage of this, besides the aforementioned failure of not identifying melting for a not-small-enough c value, is that this method might actually identify partially melted systems as just melted systems. That is, the temperature the system should fall to after melting is not fully achieved. This can be seen in Figure 8.1.

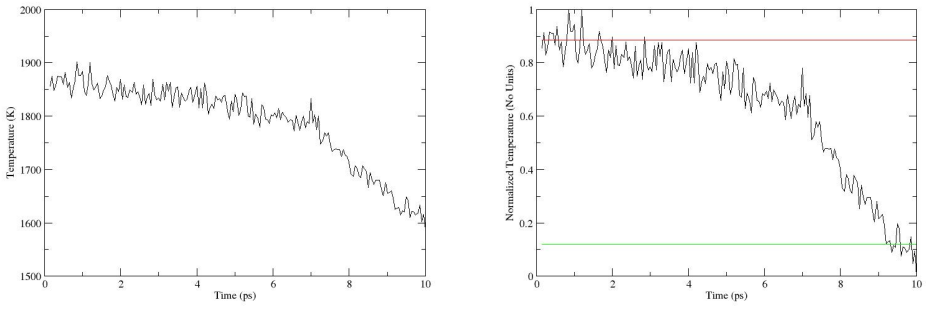

Figure 8.1: On the image in the left, the original data. On the right is the normalized data. The average for the first 10% and last 10% of the simulation for the normalized data are plotted.

Here, $T_{avg,\%p,normal} = 0.885708$ and $T_{avg,-\%p,normal} = 0.118905$, leading to a difference of 0.766803. Thus, a c value of 0.5 would classify this as melting. From the application, the CNN does not do this. The CNN does not tend to consider partially melted system as melted, which in truth is beneficial because we want to take the average temperature and pressure of the end of the simulation when it melts. A partially melted system is not very useful to us.

Finally, an optimal value for c seems to be 0.5 and one for $p\%$ should be a low value such as 10%.

An example of this method is shown below in Figure 8.2.

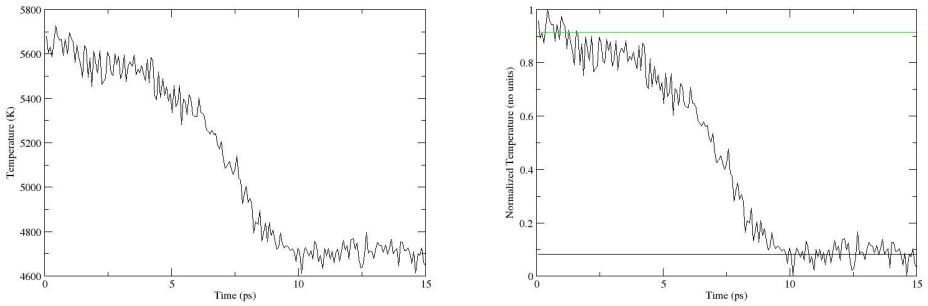

Figure 8.2: On the image in the left, the original data. On the right is the normalized data. The average for the first 10% and last 10% of the simulation for the normalized data are plotted.

Here, the two averages are 0.0818661 and 0.913804. Their difference is 0.8319379. Thus, a c value of 0.50 would identify this as melting.

One would think that taking averages such as those of Equations (8.4) and (8.3) are not necessary, for probably the last temperature value and first temperature values should suffice, but if we consider a non-melting system, this has the probability of leading to a incorrect melting classification. We can consider the system below for FCC copper at 1000 K, Figure 8.3.

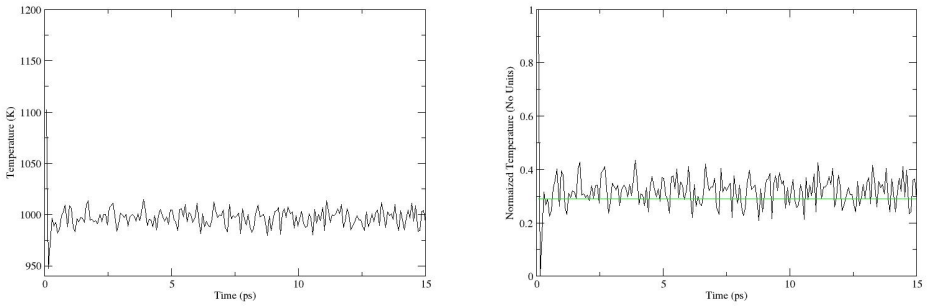

Figure 8.3: On the image in the left, the original data. On the right is the normalized data. The average in this case for $\%p$ is only the first data point and for $-\%p$ is the last data point.

Here, $T_{avg,\%p,normal} = 1$ and $T_{avg,-\%p,normal} = 0.289436$, leading to a difference of 0.710564. Thus, a c value of 0.5 would classify this as melting, which would be incorrect. This type of system, in particular, has a temperature much lower than melting, but the issue with this system is that it takes a long time to equilibrate to 1000K. The system starts off at 2000K, but it takes a small amount of time for its temperature to fluctuate about 1000K. This lag leads to its incorrect classification, and thus why taking an average is better. As seen below, we take a 10% average, and using a c value of 0.5, leads to a better classification of not melting.

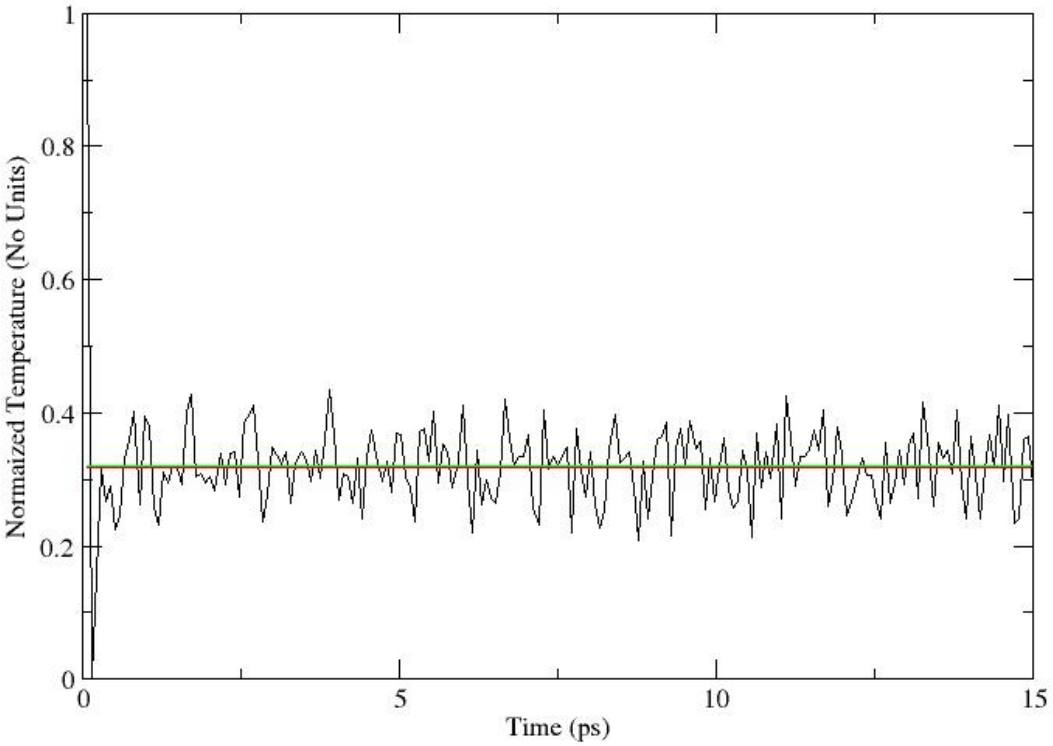

Figure 8.4: This is the normalized data. In this case, $p = 10$ and the correct classification is achieved.

Here, $T_{avg,\%10,normal} = 0.317988$ and $T_{avg,\%10,normal} = 0.321413$, yielding a difference of $-0.003425 < 0.50$.

Another advantage to note about the CNN is that a third classification of "partially melted" can be considered. As such, when this classification occurs, the simulation can be run longer and thus achieve full melting, classified by the CNN for assurance. However, this would require retraining the CNN, but in theory, the CNN would be able to do this while the numerical filter would not.

8.2 Phase Transition Lines between Crystal Structures

Although this work concerns itself with melting, the more intricate phase transitions between crystal structures is more interesting. As discussed in Subsection 2.2.2, first order phase transitions can be identified by discontinuities in the first derivative of the Gibbs free energy with respect to state variables such as entropy S and volume V. This classifies phase transitions, but one can just look for a change in a state variable to see a phase transition. After, the system can be inspected and the crystal structure can be identified. Several points at different initial states can be recorded for a phase transition, and the phase transition line can be mapped.

An example of a phase diagram can be seen in Figure 8.5.

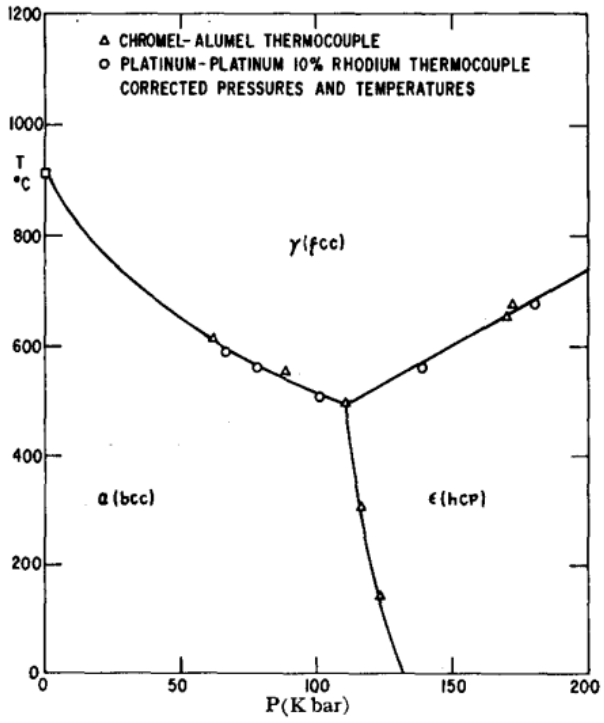

Figure 8.5: Experimental mapping of phase diagram of iron [11]. Here, we can see three regions in the P-T graph corresponding to FCC, BCC and HCP crystal structures.

For Figure 8.5, and as explained in [11], a phase change was noted by changes in electrical resistance with change of pressure or temperature. Resistance is difficult to simulate, but [11] also noted that the density of the iron sample also changed. As noted in Subsection 2.2.1, for a given lattice constant, FCC crystals are denser than SC or BCC crystals. As such, a changing density can also be used to identify a phase transition. Strategies for the calculation of phase diagrams can be explored in [37].

Given a state variable to keep track of, a machine learning method for classification like the one in this work can in theory be developed for these more intricate phase transitions between crystal structures.

8.3 Higher-Dimensional Phase Diagrams

Another subject of exploration regarding mapping the state of a system is the construction of higher-dimensional (more than 2 dimensions) phase diagrams. As noted in [12], systems such as high-entropy alloys contain multiple principal component alloys. These compounds are composed of multiple elements and thus the concentration of each element, along with some other state variable such as pressure or temperature, can be plotted. However, given e elements, the phase diagram tends to have $n = e - 1$ dimensions. As such, it is easy to go beyond the three-dimensional visualization with multiple component systems. Nevertheless, phase boundaries can still be defined using curved hyper-surfaces. A system composed of cobalt (Co), chromium (Cr), iron (Fe), nickel (Ni) and vanadium (V) is shown in Figure 8.6.

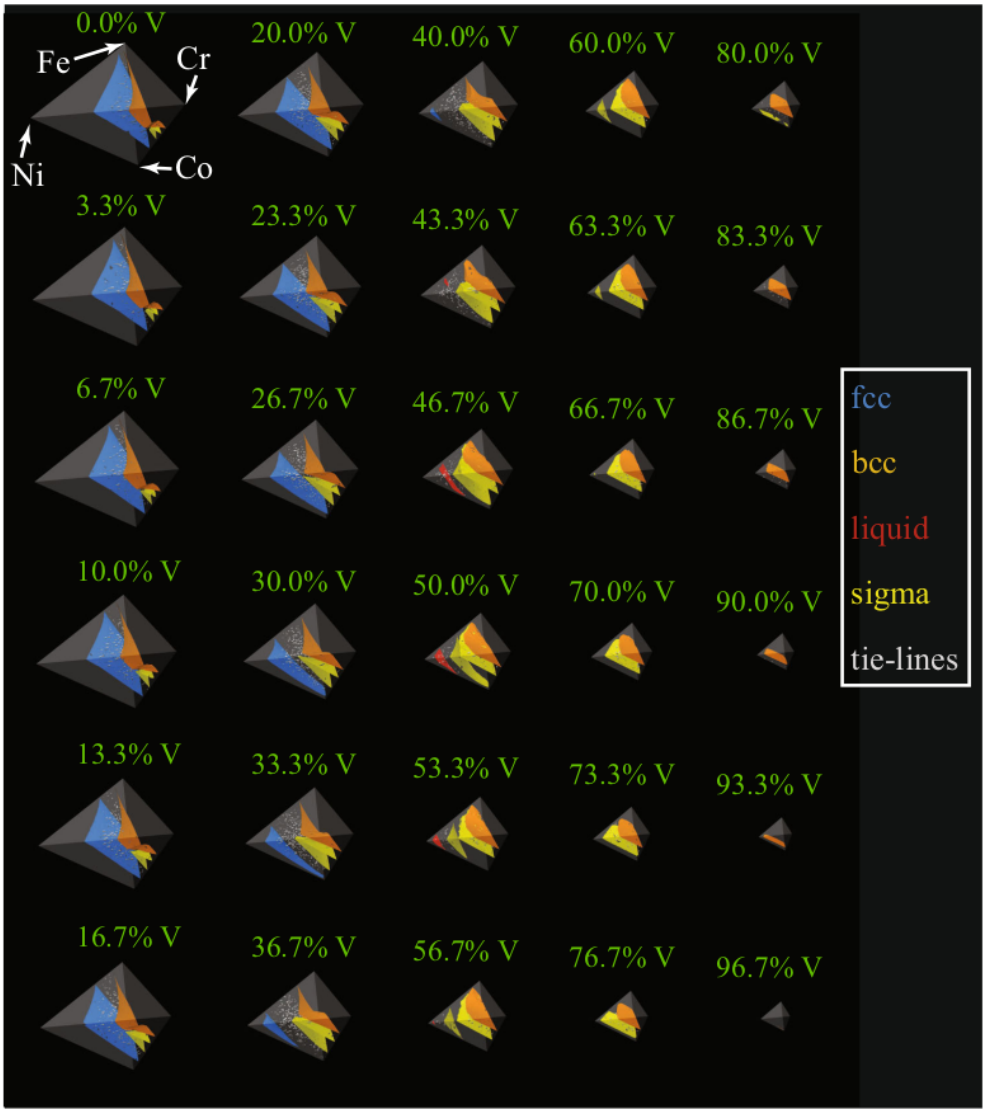

Figure 8.6: Co-Cr-Fe-Ni-V system where the surfaces indicate phase boundaries and the concentration of V is gradually increased from 0% to about 100% [12]

.

Beside particle concentration for multi-species systems, other state variables along with the two explored in this work can be varied. In particular, as discussed in Subsection 2.1.1, another extensive state variable, that was kept constant in the NVE MD simulations of this work, was volume. Thus, the exploration of mapping P-T-V phase diagrams can be

considered.

How the implementation of CNNs procedures can aid in mapping out these higher-dimensional phase diagrams is proposed for future work. Even the exploration of other machine learning methods besides NNs can be considered.

References

[1] A. B. Belonoshko, N. Skorodumova, A. Rosengren, and B. Johansson, "Melting and critical superheating,"*Physical Review B*, vol. 73, no. 1, p. 012201, 2006.

[2] F. H. Streitz, J. N. Glosli, and M. V. Patel, "Beyondfinite-size scaling in solidification simulations,"*Physical Review Letters*, vol. 96, no. 22, p. 225701, 2006.

[3] L.-F. Zhu, B. Grabowski, and J. Neugebauer, "Efficient approach to compute melting properties fully from ab initio with application to cu,"*Physical Review B*, vol. 96, no. 22, p. 224202, 2017.

[4] J. R. Morris, C. Wang, K. Ho, and C. T. Chan, "Melting line of aluminum from simulations of coexisting phases,"*Physical Review B*, vol. 49, no. 5, p. 3109, 1994.

[5] J.-P. Ansermet and S. D. Brechet,*Principles of Thermodynamics*. Cambridge University Press, Jan. 2019.

[6] F. González-Cataldo, S. Davis, and G. Gutiérrez, "Z method calculations to determine the melting curve of silica at high pressures," in*Journal of Physics: Conference Series*, vol. 720, p. 012032, IOP Publishing, 2016.

[7] M. A. Nielsen, "Neural networks and deep learning." https://www.academia.edu/download/62971418/neuralnetworksanddeeplearning20200415-115041-1t7vxpc.pdf. Accessed: 2022-9-2.

[8] M. J. Dinneen and G. Gimel'farb,*Introduction to Algorithms, Data Structures and Formal Languages*. Pearson Education New Zealand, 2009.

[9] S. Gubin, I. Maklashova, A. Selezenev, and S. Kozlova, "Molecular-dynamics study melting aluminum at high pressures,"*Physics Procedia*, vol. 72, pp. 338–341, 2015.

[10] R. Ravelo, T. Germann, O. Guerrero, Q. An, and B. Holian, "Shock-induced plasticity in tantalum single crystals: Interatomic potentials and large-scale molecular-dynamics simulations,"*Physical Review B*, vol. 88, no. 13, p. 134101, 2013.

[11] F. Bundy, "Pressure—temperature phase diagram of iron to 200 kbar, 900 c,"*Journal of Applied Physics*, vol. 36, no. 2, pp. 616–620, 1965.

[12] A. van de Walle, H. Chen, H. Liu, C. Nataraj, S. Samanta, S. Zhu, and R. Arroyave, "Interactive exploration of high-dimensional phase diagrams,"*JOM*, vol. 74, pp. 3478—-3486, 2022.

[13] D. Frenkel, B. Smit, and M. A. Ratner, "Understanding molecular simulation: From algorithms to applications,"*Phys. Today*, vol. 50, pp. 66–66, July 1997.

[14] B. L. Holian, A. F. Voter, and R. Ravelo, "Thermostatted molecular dynamics: How to avoid the Toda demon hidden in Nosé-Hoover dynamics,"*Physical Review E*, vol. 52, no. 3, p. 2338, 1995.

[15] J. Lima and A. Plastino, "On the classical energy equipartition theorem,"*Brazilian Journal of Physics*, vol. 30, pp. 176–180, 2000.

[16] G. Marc and W. McMillan, "The virial theorem,"*Advances in Chemical Physics*, vol. 58, pp. 209–361, 2007.

[17] C. Kittel and P. McEuen,*Introduction to Solid State Physics*. John Wiley & Sons, 2018.

[18] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - aflexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Comp. Phys. Comm.*, vol. 271, p. 108171, 2022.

[19] Y. Mishin, M. Mehl, D. Papaconstantopoulos, A. Voter, and J. Kress, "Structural stability and lattice defects in copper: Ab initio, tight-binding, and embedded-atom calculations,"*Physical Review B*, vol. 63, no. 22, p. 224106, 2001.

[20] G. Ciccotti, C. Dellago, M. Ferrario, E. Hernández, and M. Tuckerman, "Molecular simulations: past, present, and future (a topical issue in epjb)," 2022.

[21] D. C. Rapaport and D. C. R. Rapaport,*The Art of Molecular Dynamics Simulation*. Cambridge University Press, Apr. 2004.

[22] M. E. Tuckerman, J. Alejandre, R. López-Rendón, A. L. Jochim, and G. J. Martyna, "A Liouville-operator derived measure-preserving integrator for molecular dynamics simulations in the isothermal–isobaric ensemble,"*J. Phys. A Math. Gen.*, vol. 39, p. 5629, Apr. 2006.

[23] N. Grønbech-Jensen and O. Farago, "A simple and effective verlet-type algorithm for simulating langevin dynamics,"*Molecular Physics*, vol. 111, no. 8, pp. 983–991, 2013.

[24] H. J. Berendsen, J. v. Postma, W. F. Van Gunsteren, A. DiNola, and J. R. Haak, "Molecular dynamics with coupling to an external bath,"*The Journal of chemical physics*, vol. 81, no. 8, pp. 3684–3690, 1984.

[25] D. J. Evans and B. L. Holian, "The nose–hoover thermostat,"*The Journal of chemical physics*, vol. 83, no. 8, pp. 4069–4074, 1985.

[26] J. Lebowitz, J. Percus, and L. Verlet, "Ensemble dependence offluctuations with application to machine computations,"*Physical Review*, vol. 153, no. 1, p. 250, 1967.

[27] H. Brand, D. Dobson, L. Vo˘cadlo, and I. Wood, "Melting curve of copper measured to 16 gpa using a multi-anvil press,"*High Pressure Research*, vol. 26, no. 3, pp. 185–191, 2006.

[28] F. Simon and G. Glatzel, "Bemerkungen zur schmelzdruckkurve,"*Zeitschrift für anorganische und allgemeine Chemie*, vol. 178, no. 1, pp. 309–316, 1929.

[29] R. B. Kearfott and V. Kreinovich,*Applications of Interval Computations*. Springer Science & Business Media, Dec. 1996.

[30] A. Tanaka and A. Tomiya, "Detection of phase transition via convolutional neural networks,"*J. Phys. Soc. Jpn.*, vol. 86, p. 063001, June 2017.

[31] "R jpeg and png graphics devices."https://astrostatistics.psu.edu/su07/R/html/grDevices/html/png.html. Accessed: 20222-09-26.

[32] "Tensorflow vs pytorch — convolutional neural networks (cnn)."https://towardsdatascience.com/tensorflow-vs-pytorch-convolutional-neural-networks-cnn-dd9 Accessed: 20222-09-26.

[33] H. Kinsley, "Loading in your own data - deep learning basics with python, tensorflow and keras p.2."

[34] H. Kinsley, "Convolutional neural networks - deep learning basics with python, tensorflow and keras p.3."

[35] H. Kinsley, "How to use your trained model - deep learning basics with python, tensorflow and keras p.6."

[36] V. Zhakhovskii, N. Inogamov, Y. V. Petrov, S. Ashitkov, and K. Nishihara, "Molecular dynamics simulation of femtosecond ablation and spallation with different interatomic potentials,"*Applied Surface Science*, vol. 255, no. 24, pp. 9592–9596, 2009.

[37] H. Lukas, J. Weiss, and E.-T. Henig, "Strategies for the calculation of phase diagrams," *Calphad*, vol. 6, no. 3, pp. 229–251, 1982.

Curriculum Vitae

Christopher Ibarra worked on his B.S. in Physics from August 2015 to May 2018 at the University of Texas at El Paso (UTEP). From 2018 to 2020, he worked on his master's in physics at UTEP under the supervision of Dr. Tunna Baruah for his thesis in density functional theory calculations on electronic structures, which he defended in August 2020. After, he joined the computational science PhD program at UTEP under the supervision of Dr. Ramon Ravelo from August 2020 to December 2022 for his thesis proposal in machine learning applications to molecular dynamics simulations.