Chair of Mechanics

Master's Thesis

# Computerassisted Thermodynamics - From Gibbs energy minimizers to neural networks

Daniel Schatzl, BSc

June 2022

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

## EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Ich erkläre, dass ich die Richtlinien des Senats der Montanuniversität Leoben zu "Gute wissenschaftliche Praxis" gelesen, verstanden und befolgt habe.

Weiters erkläre ich, dass die elektronische und gedruckte Version der eingereichten wissenschaftlichen Abschlussarbeit formal und inhaltlich identisch sind.

Datum 08.06.2022

Daniel Schatzl

Unterschrift Verfasser/in
Daniel Schatzl

# Acknowledgment

First and foremost, I would like to thank my main advisor assoz.Prof. Dipl.-Ing. Dr.mont. Ernst Gamsjäger who always supported the work on this thesis with his inputs and ideas. I am however even more grateful that he supported my plans to conduct a research stay at the Colorado School of Mines in the United States of America, which led to numerous online meetings with a time shift of eight hours at both ends of the meeting.

I am also very thankful for the participation of O.Univ.-Prof. Dipl.-Ing. Dr.techn. Paul O'Leary who provided very interesting ideas on how the results and approaches could be enhanced.

Lastly, my thanks also go to the Austrian Marshall Plan Foundation, the Colorado School of Mines, and there especially to Dr. Hua Wang and Mary Cook. The Austrian Marshall Plan Foundation supported my aim to conduct research in the United States with a scholarship. I am very grateful to Dr. Wang who accepted to collaborate with me after only seeing my research proposal. He provided many valuable ideas on how the neural network implementations can be improved. Mary Cook of the International Office of the Colorado School of Mines was the one person that helped me most with my plan to come to Golden. She helped with all the formalities that occur when someone moves to a foreign country, connected me to Dr. Wang, and helped me find housing. Without her help, I probably could not have come to Mines.

# Abstract

In several real-world scenarios (e.g., in steel production), fast decisions about phases that may occur at certain temperatures and compositions in chemical systems are required. The classical approach of minimizing the Gibbs energy of the system numerically can be slow, wherefore this work aims to train and use neural networks to calculate phase equilibria for a given system. It is assumed that once trained, a neural network can provide answers quicker than the classical approach.

It is the goal of this work to approximate phase equilibria in unary and binary systems with neural networks. For this aim, Gibbs energies, entropies, enthalpies, and heat capacities of unary systems are approximated. The network for predicting binary equilibria directly uses analytical Gibbs energy functions to determine the equilibrium compositions.

Even though the approach for the unary system can be applied to approximate the Gibbs energy, entropy, enthalpy, and heat capacity functions of a unary system, it does not provide faster results for calculating phase equilibria when compared to classical methods. However, the presented approach for the binary equilibrium calculation is faster using neural networks than numerical methods. The approach using neural networks is an approximation and does not allow for exact calculations of the equilibrium compositions. It works best when applied to systems where the Gibbs energy curves do not have inflection points. This fact can be limiting for real-world applications. Therefore, an additional method to categorize phases is presented, which can classify measurements (e.g., for the heat capacity) into the phases the measurements are taken from. The presented network is trained on 78 pure elements. It can however easily be adapted to classify phases of binary systems by training it on the data of the desired systems.

Whilst the calculation of binary phase equilibria using neural networks is faster than using numerical methods, the approach has restrictions. To make fast decisions about the phases present in a system, the approach to classifying measurements rather than calculating equilibrium conditions directly is more reliable.

# Kurzfassung

In vielen realen Szenarien wie z. B. in der Stahlproduktion sind schnelle Entscheidungen darüber, welche Phasen bei bestimmten Temperaturen und Zusammensetzungen in chemischen Systemen auftreten können, erforderlich. Der klassische Ansatz, die Gibbsenergie des Systems numerisch zu minimieren, ist oft langsam. Daher ist es das Ziel dieser Arbeit, neuronale Netze zu trainieren, welche verwendet werden, um Phasengleichgewichte für ein bestimmtes System zu berechnen. Es wird davon ausgegangen, dass ein trainiertes neuronales Netz schneller Antworten liefern kann als der klassische Ansatz.

Das Ziel dieser Arbeit ist es, Phasengleichgewichte in unären und binären Systemen mit neuronalen Netzen vorherzusagen. Zu diesem Zweck werden Gibbsenergien, Entropien, Enthalpien und Wärmekapazitäten von unären Systemen approximiert. Das Netzwerk zur Vorhersage binärer Gleichgewichte verwendet direkt analytische Gibbs-Energie-Funktionen zur Bestimmung der Gleichgewichtszusammensetzungen.

Obwohl der Ansatz für das unäre System zur Annäherung an die Gibbs-Energie-, Entropie-, Enthalpie- und Wärmekapazitätsfunktionen eines unären Systems angewendet werden kann, liefert er im Vergleich zu klassischen Methoden keine schnelleren Ergebnisse für die Berechnung von Phasengleichgewichten. Der vorgeschlagene Ansatz für die Berechnung des binären Gleichgewichts mit neuronalen Netzen ist jedoch schneller als mit numerischen Methoden. Der Ansatz mit neuronalen Netzen ist allerdings eine Annäherung und ermöglicht keine exakte Berechnung der Gleichgewichtszusammensetzungen. Er funktioniert am besten, wenn er auf Systeme, bei denen die Gibbs-Energiekurven keine Wendepunkte aufweisen, angewandt wird. Dies kann für reale Anwendungen einschränkend sein. Daher wird eine zusätzliche Methode zur Kategorisierung von Phasen vorgestellt, mit der Messwerte (z. B. für die Wärmekapazität) jenen Phasen zugeordnet werden können, aus denen die Messungen entnommen wurden. Das vorgestellte Netz wurde auf 78 reine Elemente trainiert. Es kann jedoch leicht angepasst werden, um Phasen von binären Systemen zu klassifizieren, indem es mit den Daten der gewünschten binären Systeme trainiert wird.

Die Berechnung binärer Phasengleichgewichte mit Hilfe neuronaler Netze ist zwar schneller als mit numerischen Methoden, der Ansatz hat aber Grenzen. Um schnelle Entscheidungen über die in einem System vorhandenen Phasen zu treffen ist es zuverlässiger, Messungen zu klassifizieren, anstatt die Gleichgewichtsbedingungen direkt zu berechnen.

# Table of Contents

# List of Figures

# List of Tables

# Notation

In general, this work tries to stick to the common notation used in both thermodynamics and machine learning/neural networks literature. Some variable names are used in both sciences.

## Types of variables

The following notation convention is valid for both the thermodynamics and machine learning part of this work.

| | |
|---|---|
| Normal faced text – both upper- and lower-case letters | Scalar values |
| **Bold-faced text – lower-case letters** | Vectors |
| **Bold-faced text – upper-case letters** | Matrices |

## Thermodynamics

| | |
|---|---|
| $Q$ | amount of heat |
| $U$ | internal energy |
| $W$ | mechanical energy |
| $H$ | enthalpy |
| $G$ | Gibbs energy |
| $S$ | entropy |
| $C_p$ | heat capacity at constant pressure |
| $T$ | temperature |
| $p$ | pressure |
| $V$ | volume |
| $n$ | mole number |
| $x$ | mole fraction |

## Machine learning and neural networks

| | |
|---|---|
| $\boldsymbol{W}$ | Neural network weight matrix |
| $\boldsymbol{b}$ | Neural network bias vector |

| | |
|---|---|
| $\boldsymbol{x}$ | Neural network (layer) input |
| $\hat{\boldsymbol{y}}(\boldsymbol{x})$ | Neural network output given the input x |
| $\boldsymbol{s}$ | Neural network pre-activation |
| $\boldsymbol{a}$ | Neural network activation |
| $L$ | Neural network loss |

# 1 Introduction

In recent years, the terms Artificial Intelligence (AI) and Machine Learning gained immense popularity. The reason for this is simple: AI-driven solutions are changing our world and the way we live and will do so even more in the future. Banking and retail [1], applications in the life sciences, where drugs are developed, diagnostics are made or clinical trials are enhanced using AI technologies [2] or gaming like *AlphaGo,* developed by the company *DeepMind* [3], are just a few examples, where AI and Machine Learning already influence our lives.

Also in the field of thermodynamics, attempts were made to investigate the possible enhancements machine learning techniques could make. For example, *Länge* ([4]) approximated the thermodynamic functions of Gibbs energy, entropy, enthalpy, and heat capacity of different phases of pure iron using an artificial neural network. Other works (e.g., [5–7]) used neural networks to predict phase equilibria in binary systems (in form of the equilibrium composition) directly.

In this work, thermodynamic problems are investigated by means of neural networks and machine learning techniques. The main goal is to determine which phases occur at certain temperatures and compositions in a chemical system. To do so, phase equilibria must be calculated. However, calculating phase equilibria numerically can be slow. It is assumed that once trained, neural networks can make fast predictions on which phases are present.

# 2 Problem definition

The diffusion and phase transformation processes in materials are driven by the tendency to minimize a thermodynamic potential. In this case, the thermodynamic potential to be minimized is the Gibbs energy $G$ for a certain temperature $T$ at constant pressure $p$. Gibbs energy minimizers are commonly used to predict phase equilibria and chemical compositions of the equilibrated phases. These Gibbs energy minimizers have to be designed in such a way that they provide the local equilibrium conditions in each elemental cell with respect to time to describe dissipative processes like heat transfer, diffusion of components, or phase transformations. It is expected that strategies involving machine learning may help to decrease the time to compute phase equilibria. To check this assumption, equilibria are calculated in both ways, based on classical methods and using a neural network. In particular, the temperature-dependent thermodynamic functions ($G$, $S$, $H$, $C_p$) of pure substances (e.g., Fe) should be approximated based on a neural network. As a prerequisite to this task, thermodynamic data have to be compiled from the literature. These data will serve as training and testing data for the neural network. Eventually, comprehensive tests have to demonstrate that the network works well within its range of application.

Furthermore, it is planned to apply machine learning methods to calculate equilibria in binary systems, e.g., the solidus and liquidus curves in the Au-Ag system as well as in other systems. Again, the Gibbs energy has to be minimized, resulting in the equilibrium compositions that fulfill the condition of equality of the chemical potentials of the phases with respect to each component.

In addition, it is planned to classify elements and phases based on thermodynamic data.

# 3 Theoretical background and literature survey

## 3.1 Thermodynamics – a brief overview

### 3.1.1 Laws of thermodynamics

The laws of thermodynamics are essential to understanding and deriving the most important thermodynamic properties. But beyond that, the definition of temperature, the energy conservation, the introduction of entropy, and the inability to reach zero Kelvin within a finite number of steps are the foundations for a scientific description of processes occurring in our universe. For further reference, the zeroth, first, second, and third laws of thermodynamics are listed briefly in the following.

#### Zeroth law of thermodynamics

The state of thermodynamic equilibrium is characterized by uniform temperature in an isolated system. Whilst the system is not in equilibrium, there exist irreversible processes that drive the system towards equilibrium. At the equilibrium, those irreversible processes vanish and no further changes, neither physical nor chemical, occur in the system. The zeroth law of thermodynamics says, that if there exists a system A that is in equilibrium with a system B and system B is in equilibrium with a system C, then also system A is in equilibrium with system C. [8]

#### First law of thermodynamics

In [9], *Planck* formulated the first law of thermodynamics as follows [8]:
"*It is in no way possible, either by mechanical, thermal, chemical, or other devices, to obtain perpetual motion, i.e. it is impossible to construct an engine which will work in a cycle and produce continuous work, or kinetic energy, from nothing.*" [9]

Considering closed systems, the first law of thermodynamics can be formulated as [10]:

$$U_2 - U_1 = Q_{12} + W_{12} \tag{3.1}$$

Whereas indices 1 and 2 denote the transition from an initial state 1 to a final state 2.

This equation can also be written in the differential form [8]:

$$\mathrm{d}U = \mathrm{d}Q + \mathrm{d}W \tag{3.2}$$

The right side of the equation depends on sign conventions. *Moran et al* [11] for example formulate equation (*3.2*) using a – sign instead of the + sign. For consistency, this work will use the sign convention used in equations (*3.1*) and (*3.2*).

In a closed system, $\mathrm{d}W$ is equal to the mechanical work due to volume change and can be written as:

$$\mathrm{d}W = -p\mathrm{d}V \tag{3.3}$$

### Second law of thermodynamics

*Clausius* formulated the second law of thermodynamics as follows:

*"It is impossible for any system to operate in such a way that the sole result would be an energy transfer by heat from a cooler to a hotter body."* [11]

For a full cycle in a closed system and reversible processes, the second law of thermodynamics can be formulated by introducing the entropy $S$ [8]:

$$\mathrm{d}S = \frac{\mathrm{d}Q}{T}, \qquad \oint \mathrm{d}S = \oint \frac{\mathrm{d}Q}{T} = 0 \tag{3.4.1}$$

For irreversible processes, the equal sign is replaced by an inequality [8]:

$$\mathrm{d}S > \frac{\mathrm{d}Q}{T}, \qquad \oint \mathrm{d}S = 0, \qquad \oint \frac{\mathrm{d}Q}{T} < 0 \tag{3.4.2}$$

Using the *Clausius* inequality, these equations can be formulated in a very general way [11]:

$$\oint \left(\frac{\partial Q}{T}\right)_b = -\sigma_{cycle} \tag{3.5}$$

Whereas the index $b$ denotes that the integrand is evaluated on the boundary of the system. $\Sigma_{cycle}$ can take the values [11]:

| | |
|---|---|
| $\sigma_{cycle} = 0$ | reversible process |
| $\sigma_{cycle} > 0$ | irreversible process |
| $\sigma_{cycle} < 0$ | impossible |

### Third law of thermodynamics

The third law of thermodynamics is often also referred to as *Nernst heat theorem*. *Nernst* formulated the law as:

"*At the absolute zero of temperature the entropy of every chemically homogeneous solid or liquid body has a zero value."* [8]

Mathematically formulated, the third law is [8]:

$$S \to 0 \quad as \quad T \to 0K \tag{3.6}$$

## 3.1.2 Thermodynamic properties

### Entropy

The term entropy was already introduced in *3.1.1*. The change of the entropy d$S$ can be formulated by distinguishing between two different increments in the change of entropy $\mathrm{d}_eS$ and $\mathrm{d}_iS$ as:

$$\mathrm{d}S = \mathrm{d}_eS + \mathrm{d}_iS \tag{3.7}$$

Therein, $\mathrm{d}_e S$ is the system's change of entropy due to the exchange of energy and matter, and $\mathrm{d}_i S$ the change of entropy due to irreversible processes within the system. $\mathrm{d}_e S$ can take positive as well as negative values, $\mathrm{d}_i S$ on the other hand must always be equal to or greater than 0. [8]

Using equation (*3.7*) in equation (*3.4.1*) yields:

$$\oint \mathrm{d}S = \oint \mathrm{d}_e S + \oint \mathrm{d}_i S = 0$$

From the condition that $\mathrm{d}_i S \geq 0$ follows:

$$\oint \mathrm{d}_e\, S = \oint \frac{\mathrm{d}Q}{T} \leq 0 \tag{3.8}$$

This equation says that any real system that runs through a cycle of operations cannot return to its initial state without increasing the entropy of its surroundings (often referred to as the "universe"). [8]

## Enthalpy

The thermodynamic quantity $U + pV$ is a state function, specified by the state variables $U, p,$ and $V$, that often occurs in this form in thermodynamic equations. Therefore, this property was given a name: the enthalpy. [8, 11]

$$H = U + pV \tag{3.9}$$

The enthalpy is associated with an extremum principle and reaches its minimum at the equilibrium when the entropy $S$ and the pressure $p$ are kept constant. To find the minimum, the change of enthalpy must be taken into consideration:

$$\mathrm{d}H = \mathrm{d}U + V\mathrm{d}p + p\mathrm{d}V$$

As the pressure $p$ is constant when the enthalpy reaches its minimum, this term simplifies to:

$$\mathrm{d}H = \mathrm{d}U + p\mathrm{d}V$$

According to equations (*3.2*) and (*3.3*), the right-hand side of the above equation is the change of heat $dQ$.

$$\mathrm{d}H = \mathrm{d}U + p\mathrm{d}V = \mathrm{d}Q$$

From equation (*3.8*) follows, that $\mathrm{d}Q = T\mathrm{d}_e S$, which, making furtherly use of equation (*3.7*) yields: $\mathrm{d}Q = T(\mathrm{d}S - \mathrm{d}_i S)$. For the enthalpy therefore follows:

$$\mathrm{d}H = T\mathrm{d}S - T\mathrm{d}_i S$$

As for the enthalpy to reach its minimum, the total entropy $S$ is fixed ($\mathrm{d}S = 0$). Therefore, the change of enthalpy is:

$$\mathrm{d}H = -T\mathrm{d}_i S \leq 0 \tag{3.10}$$

More details can be found in the textbook by *Kondepudi* and *Prigogine* [8].

## Gibbs energy

The Gibbs energy is a thermodynamic potential, which evolves to a minimum at constant pressure $p$ and temperature $T$. It is defined as follows: [8, 10, 11]

$$G = H - TS \tag{3.11}$$

Considering the change of the Gibbs energy, its minimum can be found using the definition of the enthalpy (equation (*3.9*)):

$$\mathrm{d}G = \mathrm{d}U + p\mathrm{d}V + V\mathrm{d}p - T\mathrm{d}S - S\mathrm{d}T$$

As the pressure $p$ and temperature $T$ need to be constant for the Gibbs energy to reach its minimum, the above equation simplifies to:

$$\mathrm{d}G = \mathrm{d}U + p\mathrm{d}V - T\mathrm{d}S$$

Using equations (*3.2*), (*3.3*), and (*3.7*), this becomes:

$$\mathrm{d}G = \mathrm{d}Q - p\mathrm{d}V + p\mathrm{d}V - T\mathrm{d}_eS - T\mathrm{d}_iS$$

As $\mathrm{d}_eS = \frac{\mathrm{d}Q}{T}$, the change of the Gibbs energy can finally be written as:

$$\mathrm{d}G = -T\mathrm{d}_iS \leq 0 \tag{3.12}$$

The Gibbs energy $G$ is a minimum at equilibrium, as the irreversible processes that always decrease the Gibbs energy of the system, vanish. Details can be found in the textbook of *Kondepudi* and *Prigogine* [8].

## Heat capacity

The heat capacity is defined as the ratio of the heat absorbed by the increase in temperature:

$$C = \frac{\mathrm{d}Q}{\mathrm{d}T} \tag{3.13}$$

At constant volume respectively constant pressure, the heat capacities are called $C_V$ and $C_p$, respectively. [8]

Following equations (*3.1*), (*3.3*), and (*3.9*), the change of enthalpy is equal to the change of heat, wherefore $C_p$ can be defined as [11]:

$$C_p = \left(\frac{\mathrm{d}H}{\mathrm{d}T}\right)_p \tag{3.14}$$

## Chemical potential

In the general case of a multi-component system, the Gibbs energy is not only a function of the temperature and pressure but also of the number of moles of each component [11]:

$$G = G(T, p, n_1, n_2, \dots, n_j) \tag{3.15}$$

The Gibbs energy can then also be written as [11]:

$$G = \sum_{i=1}^{j} n_i \left(\frac{\partial G}{\partial n_i}\right)_{T,p,n_l} \tag{3.16}$$

The partial derivative $\left(\frac{\partial G}{\partial n_i}\right)_{T,p,n_l}$ is given the name chemical potential $\mu_i$ with respect to component $i$ and is defined as follows [11]:

$$\mu_i = \left(\frac{\partial G}{\partial n_i}\right)_{T,p,n_l} \tag{3.17}$$

Alternatively, this equation can be written using the activity $a_i$ of a compound $i$, which was first introduced by *Lewis* [8]:

$$\mu_i(p, T) = \mu_i(p_0, T) + RT\, ln(a_i) \tag{3.18}$$

There, $p_0$ is a known pressure value.

### 3.1.3 Thermodynamic functions derived from the Gibbs energy

#### Entropy

Substituting equation (*3.9*) into equation (*3.11*) gives:

$$G = U + pV - TS$$

The total differential d$G$ results in

$$\mathrm{d}G = \mathrm{d}U + V\mathrm{d}p + p\mathrm{d}V - T\mathrm{d}S - S\mathrm{d}T$$

Using equation (*3.7*), this can also be written as:

$$\mathrm{d}G = \mathrm{d}U + V\mathrm{d}p + p\mathrm{d}V - T\mathrm{d}_eS - T\mathrm{d}_iS - S\mathrm{d}T$$

when considering the change of the Gibbs energy. Making use of the first law of thermodynamics and specifically equations (*3.2*) and (*3.3*), $\mathrm{d}U + p\mathrm{d}V = \mathrm{d}Q$. According to the second law of thermodynamics, also $T\mathrm{d}_eS = \mathrm{d}Q$. For reversible processes, $\mathrm{d}_iS = 0$ [8]. What therefore remains is:

$$\mathrm{d}G = V\mathrm{d}p - S\mathrm{d}T$$

Taking the derivative of the above equation with respect to temperature at constant pressure gives the negative entropy, which, according to the second law of thermodynamics, is always negative (see e.g., [11]):

$$\left(\frac{\partial G}{\partial T}\right)_p = -S < 0 \tag{3.19}$$

### Enthalpy

The Gibbs-Helmholtz equation, where the enthalpy $H$ is related to Gibbs energy $G$, is derived by substituting equation (*3.19*) for the entropy in equation (*3.11*),

$$G = H + \left(\frac{\partial G}{\partial T}\right)_p T$$

and arriving at:

$$\frac{\partial}{\partial T}\left(\frac{G}{T}\right) = -\frac{H}{T^2} \tag{3.20}$$

Details can e.g. be found in the textbook of *Kondepudi* and *Prigogine* [8].

### Heat capacity

The heat capacity $C_p$ at constant pressure relates to the Gibbs energy through the following equation [10]:

$$C_p = -T\left(\frac{\partial^2 G}{\partial T^2}\right)_p \tag{3.21}$$

This can be derived from equation (*3.14*) and the Gibbs-Helmholtz equation (*3.20*):

$$C_p = \left(\frac{\mathrm{d}H}{\mathrm{d}T}\right)_p, \qquad H = -T^2 \frac{\partial}{\partial T}\left(\frac{G}{T}\right)$$

$$C_p = \left[-2T\frac{\partial}{\partial T}\left(\frac{G}{T}\right) - T^2\frac{\partial^2}{\partial T^2}\left(\frac{G}{T}\right)\right]\Bigg|_p =$$

$$= \left[-2T\left(\frac{\partial G/\partial T}{T} - \frac{G}{T^2}\right) - T^2\frac{\partial}{\partial T}\left(\frac{\partial G/\partial T}{T} - \frac{G}{T^2}\right)\right]\Bigg|_p =$$

$$= \left[2\frac{G}{T} - 2\frac{\partial G}{\partial T} - T^2\left(\frac{\partial^2 G/\partial T^2}{T} - \frac{\partial G/\partial T}{T^2} - \frac{\partial G/\partial T}{T^2} + 2\frac{G}{T^3}\right)\right]\Bigg|_p =$$

$$= -T\left(\frac{\partial^2 G}{\partial T^2}\right)_p$$

## 3.1.4 Phase equilibria

### Definition of a phase

In general, thermodynamics knows three states of matter, namely solid, liquid, and gas, which are often referred to as phases. In addition, several distinct solid, liquid, or gaseous phases can occur in a thermodynamic system. [8] A precise definition of a phase can be found in [11]:

“*The term phase refers to a quantity of matter that is homogeneous throughout in both chemical composition and physical structure. Homogeneity in physical structure means that the matter is all solid, or all liquid, or all vapor (or equivalently all gas). A system can contain one or more phases.*”

## Equilibria and equilibrium conditions

A physical system can either be in the state of equilibrium or non-equilibrium. If the system is not in equilibrium, there exist irreversible processes which evolve the system towards the state of equilibrium. These processes can be internal if the system is isolated and therefore not able to react with its surroundings or in addition also externally if the system is not isolated. Internal changes include the change of intensive properties as the temperature and the pressure toward uniform values, which occur in the state of equilibrium. Once these uniform values are reached (e.g., in the state of equilibrium, the system has uniform temperature), the system is in equilibrium and the irreversible processes driving the system towards equilibrium vanish. [8, 11]

In the state of equilibrium with constant temperature and pressure, the following must hold [11]:

$$\mathrm{d}G|_{T,p} = 0 \tag{3.22}$$

This means the Gibbs energy has an extremum at the point of equilibrium. Using the first and second laws of thermodynamics as well as the definitions for the enthalpy and the Gibbs energy, it can be shown that this extremum is a minimum in all cases. [11]

Using the chemical potential, the equilibrium condition can also be written as ([11]):

$$\mathrm{d}G|_{T,p} = \sum_{i=1}^{j} \mu_i \mathrm{d}n_i = 0 \tag{3.23}$$

## Phase equilibria in unary systems

Depending on the temperature and the pressure, different phases can be stable in a system, whereas for a phase to be stable, its Gibbs energy must be the minimum of all Gibbs energies of all phases at the conditions given. At the coexistence line p(T), two phases are present in the unary system, three phases exist at the triple point. The stability ranges of the phases are often presented in phase diagrams (see e.g., *Figure 3.1*), where the pressure is plotted against the temperature and the lines mark equilibria between two phases. Three phases coexist at the triple point. [8]

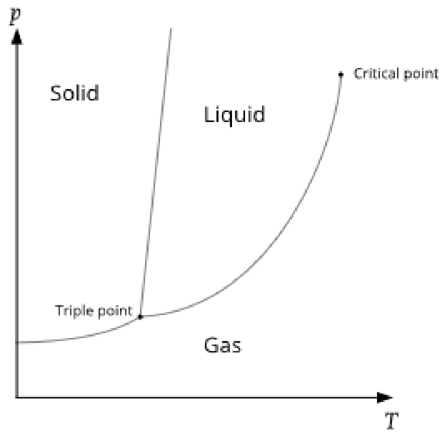


**Figure 3.1:** Phase diagram for a unary system [8]

In an equilibrium of two phases, the chemical potential of the two phases must be the same [8, 11]:

$$\mu^{\alpha} = \mu^{\beta} \tag{3.24}$$

There, the superscripts α and β denote two different phases. Equation (*3.24*) can be derived from equation (*3.23*):

$$\mu^{\alpha} \mathrm{d}n^{\alpha} + \mu^{\beta} \mathrm{d}n^{\beta} = 0$$

As the number of moles in a phase can change but not the total amount of moles in the system, the following must hold [11]:

$$\mathrm{d}n^{\alpha} = -\mathrm{d}n^{\beta}$$

Combining the last two equations leads to equation (*3.24*).

In case there are multiple phases of the same compound at one point, the chemical potentials of all phases must be the same as well because equation (*3.24*) must hold for any combination of two phases in the system. For $p$ phases, the equilibrium condition becomes, whereas in compliance with Gibbs' phase rule (number of phases + degrees of freedom = components + 2), in this case, p cannot be greater than 3: [8]

$$\mu^{\alpha} = \mu^{\beta} = \cdots = \mu^{p} \tag{3.25}$$

### Phase equilibria in binary systems

The term *mole fraction*, which is generally used to describe the thermodynamics of systems of more than one component, is introduced. The mole fraction $x_i$ is the number of moles $n_i$ of component $i$ relative to the total amount of moles $n$ in the system [11]:

$$x_i = \frac{n_i}{n} \tag{3.26}$$

A phase diagram of a binary system can, for example, plot the temperature over the mole fraction of one component. *Figure 3.2* shows a phase diagram of a mixture of two similar liquids a and b which are in equilibrium with their vapor.

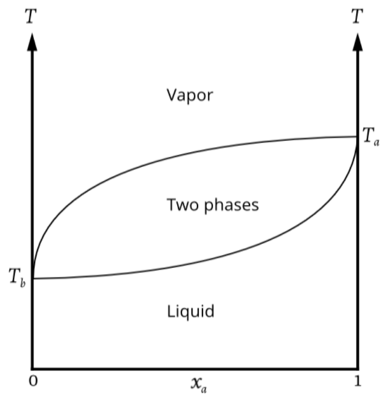


**Figure 3.2:** Phase diagram of a mixture of two liquids [8]

The phase diagram for a mixture of two solids a and b which are miscible in the liquid phase but not in the solid can be seen in *Figure 3.3.* Point E is called the *eutectic* point. [8]

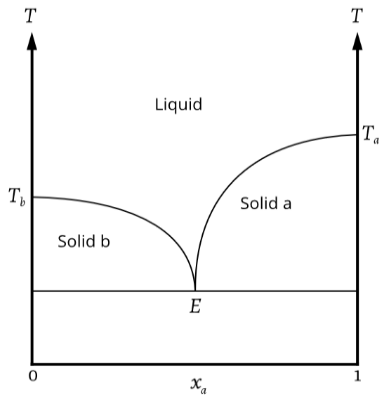


**Figure 3.3:** Solid-liquid mixture with eutectic [8]

Similar to the unary system, an equilibrium condition for the chemical potential can be defined for the binary system [11]:

$$\mu_a^\alpha = \mu_a^\beta, \qquad \mu_b^\alpha = \mu_b^\beta \tag{3.27}$$

There, the superscripts $\alpha$ and $\beta$ denote again two different phases, and the subscripts $a$ and $b$ denote two different components. This condition can, like in the unary case, be derived from the fact that the Gibbs energy of the system takes on a minimum in the equilibrium. Also, the total number of moles in the system cannot change, whereas the number of moles of different phases of a component can. [11]

The chemical potentials for a binary system can be written as [12]:

$$\mu_a^\alpha = G^\alpha - x^\alpha \frac{\partial G^\alpha}{\partial x^\alpha} \tag{3.28.1}$$

$$\mu_b^\alpha = G^\alpha + (1 - x^\alpha) \frac{\partial G^\alpha}{\partial x^\alpha} \tag{3.28.2}$$

$$\mu_a^\beta = G^\beta - x^\beta \frac{\partial G^\beta}{\partial x^\beta} \tag{3.28.3}$$

$$\mu_b^\beta = G^\beta + \left(1 - x^\beta\right) \frac{\partial G^\beta}{\partial x^\beta} \tag{3.28.4}$$

$G^\alpha$ and $G^\beta$ are the Gibbs energies of the respective phases. Using the fact that the chemical potentials of both components must be equal for both phases (equation (*3.27*)), equations (*3.28*) be simplified to:

$$G^\alpha - x^\alpha \frac{\partial G^\alpha}{\partial x^\alpha} = G^\beta - x^\beta \frac{\partial G^\beta}{\partial x^\beta} \tag{3.29.1}$$

$$\frac{\partial G^\alpha}{\partial x^\alpha} = \frac{\partial G^\beta}{\partial x^\beta} \tag{3.29.2}$$

Equation (*3.29.2*) can be derived from the equilibrium condition for component *b*. From equation (*3.27*) follows:

$$\mu_b^\alpha = \mu_b^\beta$$

Using equations (*3.28.2*) and (*3.28.4*), this can be written as:

$$G^\alpha + (1 - x^\alpha) \frac{\partial G^\alpha}{\partial x^\alpha} = G^\beta + \left(1 - x^\beta\right) \frac{\partial G^\beta}{\partial x^\beta}$$

This can be reformatted to (using equations (*3.28.1*) and (*3.28.3*)):

$$\mu_a^\alpha + \frac{\partial G^\alpha}{\partial x^\alpha} = \mu_a^\beta + \frac{\partial G^\beta}{\partial x^\beta}$$

Following equation (*3.27*) also $\mu_a^\alpha = \mu_a^\beta$ most hold and therefore equation (*3.29.2*) can be derived.

### 3.1.5 Thermodynamic Databases

For this work, different thermodynamic databases are of importance. On the one hand, this is the *FactSage* software presented in [13], and on the other hand also the *SGTE data for pure elements* of [14]. Given the data of each of the two databases, the minimum Gibbs energy for a certain temperature and pressure can be calculated. Following this, phase equilibria and phase transitions can be determined.

The *FactSage* software provides extensive amounts of data not only for pure elements but also for compounds. For pure elements and compounds, the Gibbs energy, entropy, enthalpy, and heat capacity over a user-specified temperature range can be retrieved. The data comes in tabular form and its source for pure elements is the aforementioned *SGTE data for pure elements* [13]. The sources for other data in the *FactSage* software can also be found in chapter 3 of [13].

The *SGTE data for pure elements* represents the Gibbs energy, entropy, enthalpy, and heat capacity for 78 elements in the form of functions with constant coefficients in certain temperature ranges. As the enthalpy and therefore the Gibbs energy have no absolute value, a reference state of 298.15K and a pressure of 1 bar is used for the tabulation of the enthalpy of formation. The Gibbs energy functions are tabulated using the enthalpy at the reference state and the absolute values for the entropy. [14]

The functions for the Gibbs energy, entropy, enthalpy, and heat capacity are the following [14]:

$$G = a + b \cdot T + c \cdot T \cdot \ln(T) + \sum d \cdot T^n$$

$$S = -b - c - c \cdot \ln(T) - \sum n \cdot d \cdot T^{n-1}$$

$$H = a - c \cdot T - \sum (n-1) \cdot d \cdot T^n$$

$$C_p = -c - \sum n \cdot (n-1) \cdot d \cdot T^{n-1}$$

In these equations, $T$ stands for the temperature, $n$ are integer values (often 2, 3, or -1, but also other values are possible and occur). The coefficients $a, b, c$, and $d$ (one $d$ value for every unique $n$) are listed in the data for different temperature ranges. For elements with magnetic properties like Fe, an additional magnetic term is added to each of the functions. Also, pressure-dependent terms are added to the functions of some elements. [14]

*Barin* [15] presents thermodynamic tables not only for pure elements but also for compounds. For pure elements, the values slightly differ from the ones obtained from evaluating the *SGTE* equations. Therefore, these tables are used to simulate real measurement data and therefore test the performance of the classification network on data that does not come from the *SGTE* database.

Phase equilibria (e.g., solidus/liquidus curve in the Ag/Au-system) are calculated by using thermodynamic data and the resulting coexistence curves are compared to those obtained by the software *ThermoCalc* [16].

## 3.2 Basics of neural networks and machine learning

### 3.2.1 Types of machine learning

Machine Learning is a technology that over the past two decades has become an important field in computer science [17]. In general, it is a computer system that is programmed to learn the theoretical laws of any learning system automatically [18]. It has a very broad spectrum of applications, including web page ranking for search engines, automatic translations, classification of face images, speech recognition, and many more [17]. Machine Learning is often divided into three subcategories: supervised, unsupervised, and reinforcement learning [19].

This work will mostly use supervised learning techniques.

#### Supervised learning

Supervised machine learning focuses on labeled data. This means, that for every data point, a correct label exists, which is often assigned to it by humans. The machine learning model's task is to correctly learn and predict the label for a data point when only given its features. [19]

Common supervised learning tasks are classification and regression. In classification, the model learns to classify the input into finite many classes. Regression handles continuous data, whereas the goal often is to approximate functions from the given data points. [20]

#### Unsupervised learning

As opposed to supervised learning, which needs labeled data, unsupervised learning does not work with labeled data. It tries to find a good model by just knowing the structure of the data, but nothing else. [19]

Common unsupervised learning tasks are clustering, density estimation, or down-projection. In clustering, as the name suggests, the goal is to divide the data into clusters where data points are similar. Density estimation tries to determine the data distribution in the input space. Down-projection aims to reduce the dimensionality of multi-dimensional data into a space, where the data can be visualized. [20]

#### Reinforcement learning

Like in unsupervised learning, also in reinforcement learning there are no labels or target values given. It rather has a so-called agent in a defined environment which can perform certain actions which change its state. Based on the agent's current state, it tries to learn the best possible action for this state. Usually, a reward signal is used to reward or punish the agent for good or bad actions and tries to use these learnings to make good predictions in the future. [19]

Reinforcement learning is used to learn to play games, for example, Backgammon [20] or the Chinese board game *Go*, as the company *DeepMind* showed with *AlphaGo* which was able to beat the best *Go* players in the world [3]. Also, autonomous driving often relies on reinforcement learning [19].

### 3.2.2 General idea and types of neural networks

Gurney gives a good definition of neural networks:

"*A neural network is an interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the interunit connection strengths, or weights, obtained by a process of adaptation to, or learning from, a set of training patterns.*" [21]

This definition is very general and therefore allows for different kinds of neural networks. There are many different types of neural network architectures in existence, whereas for this work, (deep) feed-forward networks will be most important. Therefore, only the idea and theory behind (deep) feed-forward networks are presented in this chapter.

## Basic structure of feed-forward neural networks

In easier words, the definition of neural networks in [21] from above says that a neural network can take an input and give an output that is adapted to the desired task. The network is made up of elements called either units, nodes, or also neurons (this work uses the term node(s)), which are structured in layers. Each node's input vector is multiplied with a weight matrix before being added up. Adding a bias term ([20]) to this sum gives the pre-activation, which is passed through a so-called activation function $f$ to build the node's output. This output can either be the output of the network or serve as an input for the next layer. *Figure 3.4* shows a visualization of an arbitrary node. [21]

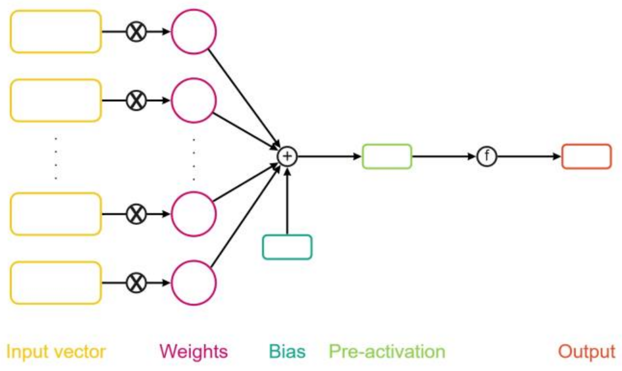


**Figure 3.4**: A node in a neural network [4]

*Figure 3.5* shows an arbitrary network where each circle marks a node in the network. It consists of three layers: the input, one hidden, and the output layer. Whilst every network has an input and an output layer, it can have an arbitrary number of hidden layers. Networks with more than 10 hidden layers are often referred to as deep networks [19]. Also, the number of nodes in a layer is not restricted and can be as small as one node per layer but also be a large collection of nodes [21].

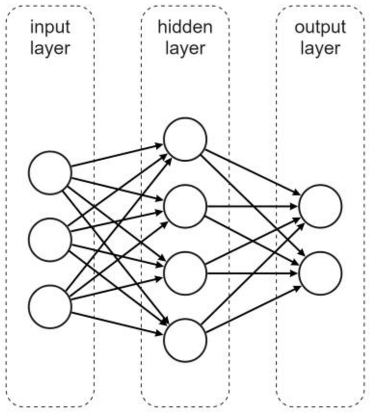


**Figure 3.5:** Feed forward neural network [19]

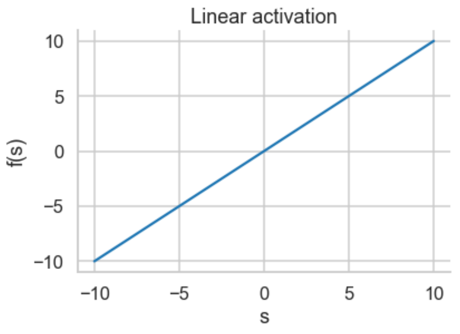
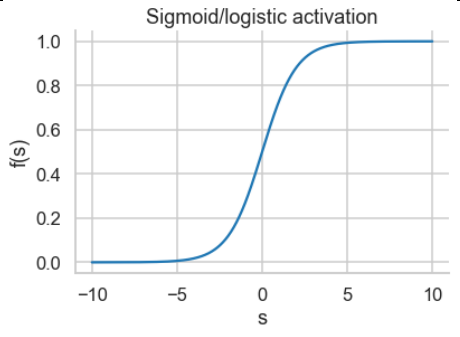
## Mathematical representation of neural networks

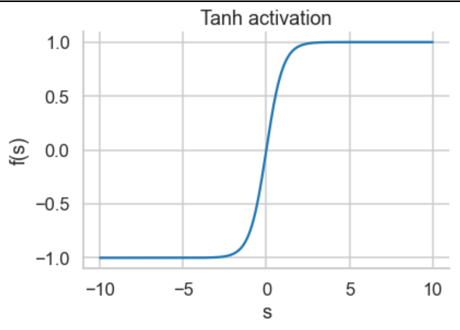
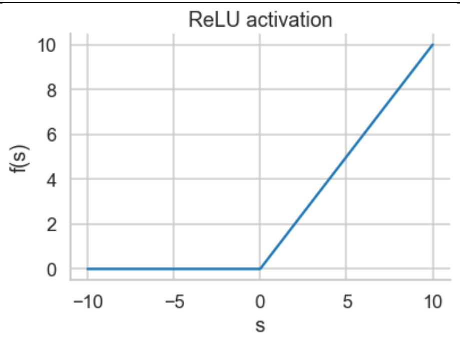
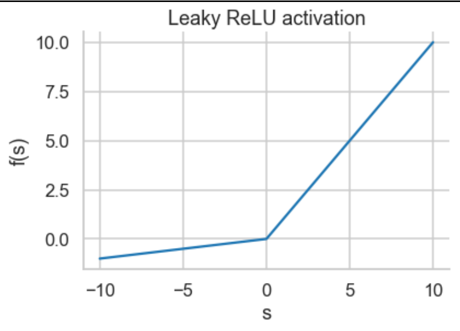
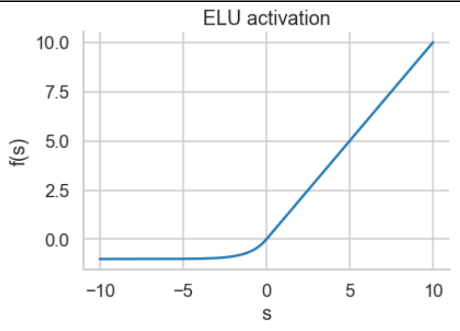
The above chapter gives a very abstract idea of how neural networks are built. Mathematically, neural networks can be formulated similarly to [20]. For an arbitrary node $j$ in an arbitrary layer of a neural network with an input vector $\boldsymbol{x}$ of length $D$, the pre-activation $s_j$ can be formulated as:

$$s_j = \sum_{i=1}^{D} w_{ji}\, x_i + b_i \tag{3.30}$$

Where $s_j$ is the entry in the $j$th row of the pre-activation vector $\boldsymbol{s}$, $w_{ji}$ is the entry in the $j$th row and $i$th column of the weight matrix $\boldsymbol{W}$, $x_i$ the entry in the $i$th row of the input vector $\boldsymbol{x}$ and $b_i$ the entry in the $i$th row of the bias vector $\boldsymbol{b}$. $\boldsymbol{s}$ is called pre-activation because, in the next step, the output of this matrix-vector multiplication is passed through a (typically non-linear) function called activation function so that it can represent an extremely large number of prediction mappings [19]. Without a non-linear activation function, a neural network just represents a linear transformation and would not be able to learn complex tasks [22]. In general, an activation function can be any function. If the layer is the output layer, depending on the task, different activation functions are used: for regression problems, the output activation is typically the linear function, for binary classification, it is the logistic sigmoid function and for multi-class classification, a softmax activation is used [20]. In hidden layers, also other functions are common as activation functions. *Table 3.1* gives an overview of the most used activation functions.

**Table 3.1:** Common activation functions [22]

| Name | Function | Graph |
|---|---|---|
| Linear | $f(s) = s$ |  |
| Sigmoid or Logistic | $f(s) = \dfrac{1}{1+e^{-s}}$ |  |

| | | |
|---|---|---|
| Tanh | $f(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$ |  |
| ReLU | $f(s) = \max(0, s)$ |  |
| Leaky ReLU | $f(s) = \max(0.1 \cdot s, s)$ |  |
| ELU | $f(s) = \begin{cases} s, & s \geq 0 \\ \alpha(e^s - 1), & s < 0 \end{cases}$ |  |
| Softmax | $f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$ | |

Applying an activation function $f$ to the pre-activation $\boldsymbol{s}$ gives the activations $\boldsymbol{a}$:

$$a_j = f(s_j) = f\left(\sum_{i=1}^{D} w_{ji}\, x_i + b_i\right) \tag{3.31}$$

Where $a_j$ is the entry in the $j$-th row of the activation vector $\boldsymbol{a}$. The activation can now be either a network output or an input to the next layer. In case it is the input to the next layer, the next layer's activation looks as follows when the superscripts 1 and 2 denote the number of the layer and $M$ is the length of the activation vector of the first layer [20]:

$$a_k{}^{(2)} = f^{(2)}\left(\sum_{j=1}^{M} w_{kj}^{(2)}\, a_j^{(1)} + b_k^{(2)}\right) = f^{(2)}\left(\sum_{j=1}^{M} w_{kj}^{(2)}\, f^{(1)}\left(\sum_{i=1}^{D} w_{ji}^{(1)}\, x_i + b_i^{(1)}\right) + b_k^{(2)}\right) \tag{3.32}$$

There, $a_k$ is the entry in the $k$-th row of the activation vector $\boldsymbol{a}$ of the second layer. This procedure is continued for every layer there is in the network.

### 3.2.3 Training a neural network

The goal of a neural network is to find weight matrices $\boldsymbol{W}$ and bias vectors $\boldsymbol{b}$ that allow the network to approximate the given target with its output as well as possible. To measure how well the network can do this, loss functions or also called cost functions are used. The loss function gives a scalar value that is small when the network can approximate the target data well and big when not. To adjust the network parameters so that this loss is minimized, an algorithm called gradient descent is used. [23]

It depends on the task, which loss function is used. Even for one task, multiple loss functions can be applied. For regression tasks, it is common to use the mean squared error loss, where $\boldsymbol{y_i}$ denotes the target vector:

$$L(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n}\big(y_i - \hat{y}_i(x_i)\big)^2 \tag{3.33}$$

In the case of classification, the cross-entropy loss is usually used:

$$L(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x}) = -\sum_{i=1}^{C} y_i \log\big(\hat{y}_i(x_i)\big) \tag{3.34}$$

There are also other loss functions, these two are however the most commonly used. [24]

To now update the network parameters $\boldsymbol{W}$ and $\boldsymbol{b}$**,** the afore-mentioned gradient descent algorithm is used. It calculates the partial derivatives of the network's loss with respect to every single network parameter (i.e., every entry in every weight matrix and every bias vector in the network). The partial derivatives are multiplied with a hyperparameter called learning rate $\eta$ and subtracted from the original parameter. For an arbitrary weight vector $\boldsymbol{w_k}$ part of a weight matrix, the update looks as follows when the updated weight vector is $\boldsymbol{w_k'}$:

$$\boldsymbol{w_k'} = \boldsymbol{w_k} - \eta\frac{\partial L}{\partial \boldsymbol{w_k}} \tag{3.35}$$

For the bias vector, this works analogously. [23]

This is the most basic update rule, but there are also more sophisticated methods to update neural network parameters. The update rules, which are often also referred to as optimizers, that are used in this work and deviate from this basic rule will be introduced in the description of the respective implementation.

The update of the parameters is an iterative process. This means every input is used multiple times. Between seeing the same input, at least one update (often multiple updates) is made to the network, leading in the best case to a minimum loss [20]. In this work, one iteration through all inputs is called an epoch. The number of times the network receives the same input is therefore called the number of epochs, which is a parameter to be chosen optimally for each training routine.

To speed up the training process, the inputs are often passed to the network in batches. Such a batch can contain all input values, whereas the network is evaluated on all input values at once. More commonly used are batches that contain only parts of the input values at once. [20] For this work, the number of values in one batch will be referred to as the batch size.

### 3.2.4 Generalization and dataset splits

The main goal when training a neural network is to be able to generalize the predictions it makes. This means the network should also be able to make accurate predictions for input data it has not been trained on. To test the generalization abilities, the dataset is split into a training and a test set, whereas both sets are generated using the same procedure. During the training phase, the network is trained using the training set whereas after that its generalization abilities are tested using the test set. In some cases, also a second split and a so-called validation set are used. The validation set is used to compare different sets of network parameters and to choose the set which gives the best performance. [20]

### 3.2.5 Neural networks in Python

The programming language *Python* is one of the most used languages to implement neural networks. Standard libraries for this purpose are *Tensorflow* [25] and *Pytorch* [26]. Both allow the programmer to easily and efficiently implement the concepts presented above. For the implementation of this work, *Pytorch* is used to build neural networks and training routines.

## 3.3 Applications of neural networks to thermodynamics

As neural networks and machine learning gained popularity over the last years, experiments in all kinds of fields were made to apply those technologies. Also in thermodynamics, work was done to discover possible applications. In the following, recent publications regarding the topic of neural networks and machine learning in thermodynamics are presented.

## 3.3.1 Neural networks applied to unary systems

In [4], *Länge* presented an approximation of the thermodynamic functions $G(T), S(T), H(T) - H(T_{ref})$ & $C_p(T)$ of unary systems in a temperature range from $0K - 6000K$ using an artificial neural network. The paper presents the results of the specific application of the approximation of above-mentioned functions for pure iron. [4]

*Länge* used a network consisting of two subnetworks, where each has one hidden layer (*Figure 3.6*). Whilst the hidden layer of subnetwork 1 (purple in the figure) has only one node, no specifications are made about the number of nodes in the second subnetwork (orange in the figure). The networks take as input ($x^t$) temperature values (which can be different for the four functions) and output an approximation for the Gibbs energy $G(T)$. For the other functions, *Länge* made use of the partial differential equations presented in *3.1.3,* which link the entropy, enthalpy, and heat capacity to the Gibbs energy. Using equations *3.31* and *3.32*, the network's parameters can be re-used to approximate the derivatives of the Gibbs energy. As the network's input $x^t$ are temperature values and the networks output are Gibbs energy values, the partial derivative of the output (e.g., equation *3.32*) with respect to the input $x^t$ can be taken. Taking the derivatives according to the equations presented in *3.1.3* gives the entropy, enthalpy, and heat capacity values as functions of the network parameters. This can be interpreted as having one network per property but with shared parameters. Therefore, it would be possible to train a network only on the data of one function and receive the network for the others at the same time. [4]

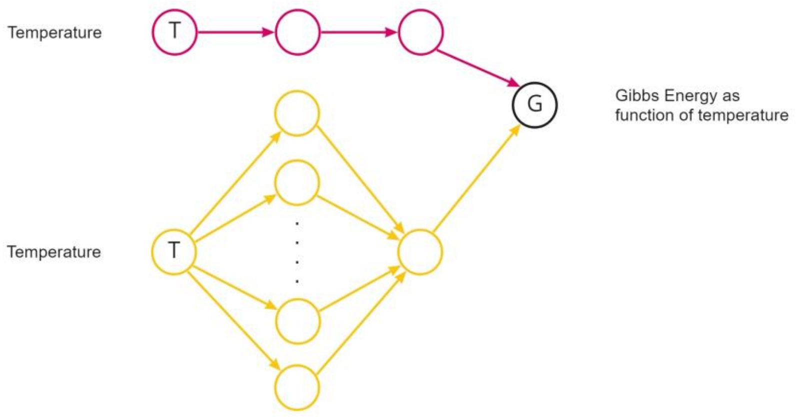


**Figure 3.6:** Network architecture [4]

Both subnetworks use different activation functions. For the first subnetwork, a function that aims to introduce laws of physics into the network is used [4]:

$$f_a(s) = E_0 + \frac{3}{2}R\theta_E + 3Rs\log\left(1 - e^{-\theta_E/s}\right) - \frac{1}{2}as^2 - \frac{1}{6}bs^3 \tag{3.36}$$

There, $R$ is the universal gas constant, $s$ are the pre-activations and all the other variables are network parameters that are learned by the network. The second subnetwork uses the softplus function as its activation function [4]:

$$f_b(s) = \log(e^s + 1) \tag{3.37}$$

As a loss function, *Länge* used the mean squared error for each of the four thermodynamic functions and summed them up. The magnitudes of the functions' values vary, therefore the contribution of each function to the total loss is weighted by undisclosed factors $q$ :

$$L_{\text{total}} = q_{\text{G}}L_{\text{G}} + q_{\text{S}}L_{\text{S}} + q_{\text{H}}L_{\text{H}} + q_{\text{C}}L_{\text{C}} \tag{3.38}$$

Where $L_{\text{G}}, L_{\text{S}}, L_{\text{H}}$ and $L_{\text{C}}$ are the mean squared errors for the respective functions. [4]

The work led to good results. E.g., the heat capacity of the bcc phase of pure iron could be approximated well in the range from $0K$ to $3000K$. Further results can be found in the original paper.

Based on *Länge's* network, the present work investigates the advantages and disadvantages of a different network architecture for approximating the Gibbs energy, entropy, enthalpy, and heat capacity in unary systems.

### 3.3.2 Neural networks applied to binary systems

While the paper ([4]) presented above tried to approximate thermodynamic functions in unary systems, also several papers were published on the prediction of phase equilibria in binary systems using neural networks. Many of the networks for phase equilibria prediction found in the literature are restricted to certain systems or elements: The network presented by *Kan et al* [5] can predict phase equilibria in aqueous two-phase systems, and *Farzi et al* [7] estimated the phase equilibria of eleven binary systems containing acetone. *Bilgin et al* predicted the vapor-liquid equilibria of six systems of different chemical structures in [6].

#### Goals, approaches, and network architectures

All the above-mentioned works [5–7, 27] used feed-forward neural networks. As the networks are applied to different chemical systems, both the number of input and output nodes are different. However, all of them predict a fixed number of equilibria composition values.

*Farzi et al* [7] used literature data for the vapor-liquid equilibria of eleven binary systems containing acetone in the temperature range from 298.15-391.25K and the pressure range from 2.640-101.33kPa. Amongst the eleven binary systems were e.g., benzene, ethanol, or ethyl acetate. The network used in this paper consists of one hidden layer, an input layer with five nodes, and an output layer with 2 nodes. The predictions made are the mole fractions in the equilibrium of acetone in liquid and vapor phases. As input, the network receives critical temperature, critical pressure, acentric factor, temperature, and pressure values. The best results were achieved with a network consisting of 19 nodes in the hidden layers and a logarithmic sigmoid function as the activation function of the hidden layer and a linear activation in the output layer. The dataset was split so that 70% of the data was used for training, 20% for validation, and 10% for testing. For the majority of the eleven systems used, the phase equilibria and as a result, the phase diagrams could be predicted well and with only small deviations from the true values. [7] For the present work, the network presented in [7] cannot be applied directly because it only has two output nodes and can therefore only predict one equilibrium (i.e., two equilibrium composition values). For a network to be able to predict equilibria also for other systems, where there can occur multiple equilibria at a given temperature, more than two output nodes are required.

The work by *Kan et al* [5] aimed to predict equilibria in aqueous two-phase systems. It compared two different network architectures. Both networks have three input and four output nodes. The difference is that one network is fully connected, whereas the other network is made up of four subnetworks. For both networks, the inputs are physical properties (weight proportions and molecular weights of the components) which are the base for predicting the equilibrium compositions in the 4 nodes of the output layer. The layers are activated by the sigmoid function. [5] Similar to the work by *Farzi et al* [7], also these networks cannot be directly applied in the present work as both networks have 4 output nodes and are therefore restricted in the number of equilibria that can be predicted at once.

In [6], the goal of *Bilgin et al* was to predict both equilibrium compositions as well as activity coefficients for six different systems. It used a fully connected feed-forward neural network with two hidden layers and the logarithmic sigmoid function as activation. The predictions were made given the low boiling component concentrations in the liquid phase, whereas the network's outputs are the concentrations in the vapor phase as well as the activity coefficients of the liquid phase. For this aim, the network has three nodes in the output layer and can therefore only predict the composition values of one equilibrium. [6] This fact is again limiting when trying to predict a variable number of equilibria in different chemical systems, wherefore also this approach cannot be applied directly in the present work.

# 4 Methods and implementations

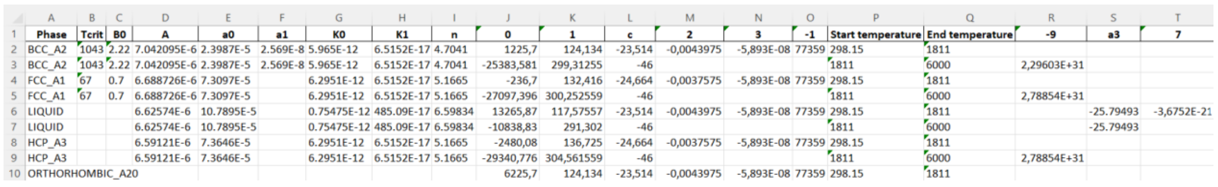
For this work, multiple *Python* packages are implemented. Each of the tasks is structured in a single package and additionally, a handler for the *SGTE* data, which is used to create the training, testing, and validation datasets, is also structured in a package. In total, this leads to five packages which are explained in this chapter:

- **sgte:** data handler to create the training, testing, and validation datasets
- **laenge:** rebuild of the network proposed by *Länge* (see *3.3.1*)
- **thermonet: a** different approach to what was presented in the paper by *Länge*
- **binarypredictor:** neural networks that predict phase equilibria in binary systems
- **thermoclassifier: a** classifier that, given measurement data for the heat capacity of any of 78 elements, predicts both the element and the phase(s) of the measurements

## 4.1 SGTE data handling – sgte

### 4.1.1 Data extraction and ordering

As described in *3.1.5,* the *SGTE* data is listed in [14] as coefficients of functions. Using a *Python* script, these coefficients and in the case of the polynomial terms, also the corresponding exponents, are extracted from the *SGTE* PDF file and stored in one *Excel* file per element. The *Excel* files are stored inside the package so that the package can be used on any computer. *Figure 4.1* shows for the example of iron, how these files are structured.



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **Phase** | **Tcrit** | **B0** | **A** | **a0** | **a1** | **K0** | **K1** | **n** | **0** | **1** | **c** | **2** | **3** | **-1** | **Start temperature** | **End temperature** | **-9** | **a3** | **7** |
| 2 | BCC_A2 | 1043 | 2.22 | 7.042095E-6 | 2.3987E-5 | 2.569E-8 | 5.965E-12 | 6.5152E-17 | 4.7041 | 1225,7 | 124,134 | -23,514 | -0,0043975 | -5,893E-08 | 77359 | 298.15 | 1811 | | | |
| 3 | BCC_A2 | 1043 | 2.22 | 7.042095E-6 | 2.3987E-5 | 2.569E-8 | 5.965E-12 | 6.5152E-17 | 4.7041 | -25383,581 | 299,31255 | -46 | | | | 1811 | 6000 | 2,29603E+31 | | |
| 4 | FCC_A1 | 67 | 0.7 | 6.688726E-6 | 7.3097E-5 | | 6.2951E-12 | 6.5152E-17 | 5.1665 | -236,7 | 132,416 | -24,664 | -0,0037575 | -5,893E-08 | 77359 | 298.15 | 1811 | | | |
| 5 | FCC_A1 | 67 | 0.7 | 6.688726E-6 | 7.3097E-5 | | 6.2951E-12 | 6.5152E-17 | 5.1665 | -27097,396 | 300,252559 | -46 | | | | 1811 | 6000 | 2,78854E+31 | | |
| 6 | LIQUID | | | 6.62574E-6 | 10.7895E-5 | | 0.75475E-12 | 485.09E-17 | 6.59834 | 13265,87 | 117,57557 | -23,514 | -0,0043975 | -5,893E-08 | 77359 | 298.15 | 1811 | | -25.79493 | -3,6752E-21 |
| 7 | LIQUID | | | 6.62574E-6 | 10.7895E-5 | | 0.75475E-12 | 485.09E-17 | 6.59834 | -10838,83 | 291,302 | -46 | | | | 1811 | 6000 | | -25.79493 | |
| 8 | HCP_A3 | | | 6.59121E-6 | 7.3646E-5 | | 6.2951E-12 | 6.5152E-17 | 5.1665 | -2480,08 | 136,725 | -24,664 | -0,0037575 | -5,893E-08 | 77359 | 298.15 | 1811 | | | |
| 9 | HCP_A3 | | | 6.59121E-6 | 7.3646E-5 | | 6.2951E-12 | 6.5152E-17 | 5.1665 | -29340,776 | 304,561559 | -46 | | | | 1811 | 6000 | 2,78854E+31 | | |
| 10 | ORTHORHOMBIC_A20 | | | | | | | | | 6225,7 | 124,134 | -23,514 | -0,0043975 | -5,893E-08 | 77359 | 298.15 | 1811 | | | |

**Figure 4.1:** *Excel* table for the *SGTE* coefficients of iron (Fe)

The column "Phase" stores the name of the phase for which the coefficients are listed in the corresponding row. Following this, columns B to I and column S list coefficients for the magnetic and pressure-dependent terms. These coefficients are only stored for those elements, where either a magnetic or pressure-dependent term or both are used in the *SGTE* equations. The columns with numeric values as column names refer to the polynomial terms, whereas the column name, in this case, is the exponent of the respective polynomial term. In column L, the *c* value is listed, which refers to the coefficient of the logarithmic term in the *SGTE* equation of the Gibbs energy. The columns "Start temperature" and "End temperature" define the temperature ranges in Kelvin in which the coefficients of the respective rows are valid.

As the coefficients are extracted from a PDF file automatically using a *Python* script, the ordering of the columns is made based on the first occurrence of the column name in the PDF file. *Figure 4.1* therefore only gives an example of how these files are structured. For other elements, the table orderings can look different. When using the tables to create the training, testing, and validation datasets, the columns are referenced by the column name, which is why a certain ordering is not required.

### 4.1.2 Dataset creation

Using the class SGTEHandler, the Gibbs energy, entropy, enthalpy, or heat capacity values of any element and any phase in the *SGTE* data can be calculated for any temperature in the defined ranges.

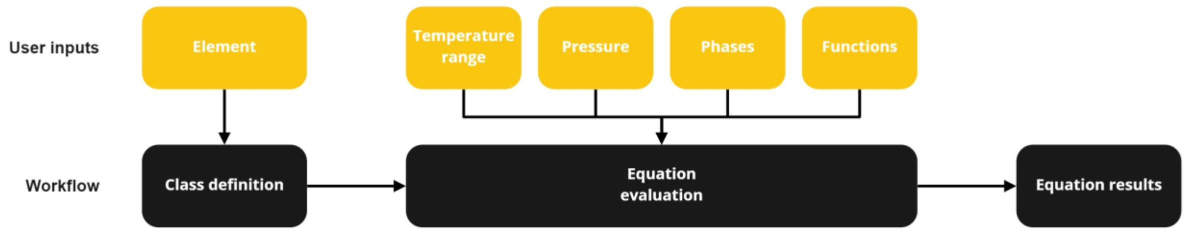


**Figure 4.2:** Basic workflow of the SGTEHandler

An SGTEHandler is always defined for one element which can either be provided as an argument when defining the class or as a command-line input. For the given element, the corresponding *Excel* file (see *4.1.1*) is loaded as a *pandas DataFrame*. To evaluate the equations, the corresponding class method must be called, whereas the following arguments must be passed:

- **Temperature range and temperature step:** A start and end value for the temperature range as well as the size of the steps made in this range need to be defined in Kelvin. For the input, the ranges in which the coefficients are defined in the *SGTE* data do not need to be known. The class handles transitions between the defined ranges automatically and returns smooth functions. Also, the minimum and maximum values of the defined ranges do not necessarily have to be known as the method only considers temperatures for which the coefficients are defined.
- **Pressure:** A pressure value needs to be passed as an argument as a float value. For most of the elements, the *SGTE* equations are only defined for a pressure of $1bar$. Therefore, for these elements entering different pressure values will not affect the results.
- **Phases:** The phases for which the evaluation should be made need to be defined and passed as a list of strings, whereas these strings can either be the phase names as defined in the *Excel* coefficient data sheet or “all”. In the first case, only the phases provided are considered, in the second case all phases for which coefficients are defined are taken into account.
- **Functions:** For each of Gibbs energy, entropy, enthalpy, and heat capacity, a Boolean value has to be passed to determine which functions to evaluate.

For the equation evaluation, the coefficients from the *Excel* file are used to solve the equations. The values of the polynomial terms are calculated by using the column names as the exponents of the temperature (for the equations see *3.1.5*) and multiplying them with the respective coefficients. If magnetic and pressure-dependent terms are defined for an element, also these equations are solved for every temperature in the range as well as the logarithmic term. For the output, the three results are added to each other. As a result of the evaluation process, a *pandas DataFrame* containing all the values is stored as a class attribute.

### 4.1.3 Stable properties

The process mentioned in *4.1.2* does not consider whether the phase considered is stable at a temperature and pressure or not. As in some cases, it is necessary to only get the values for any of the functions for only stable phases, a class method is introduced to return those values. For a stable phase, its Gibbs energy must be the minimum Gibbs energy of all possible phases at the given conditions (see *3.1.4*). Therefore, to get the values of the stable phases, only the function values of a phase for which the Gibbs energy is smaller than the Gibbs energy of all the other phases are returned.

For visualization purposes, it is possible to plot any of the four functions over the temperature only for stable phases. The phase information can be included in the plot so that it can be seen which phase is stable at a given temperature. *Figure 4.3* shows as an example the Gibbs energy function of iron over the temperature in the range from $300K$ to $6000K$, whereas the different coloring shows the stable phases at a temperature.

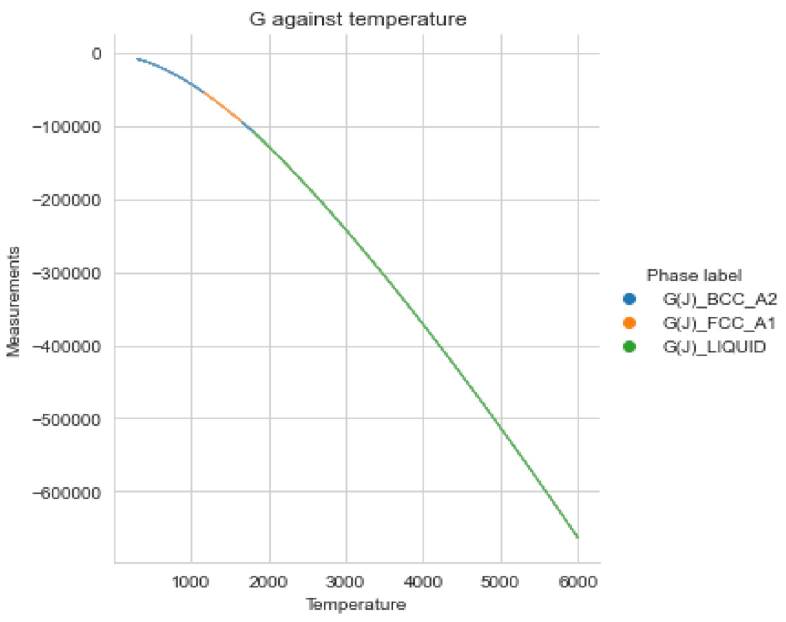


**Figure 4.3:** Gibbs energy of iron over the temperature

## 4.2 Adaption of the Länge network – laenge

### 4.2.1 Goal and workflow

In this package, the network presented by *Länge* in [4] is implemented and trained to check how well and easy the network can be trained so that it can serve as a basis for other networks in this work. *Länge* uses experimental data to approximate the thermodynamic functions of Gibbs energy, entropy, enthalpy, and heat capacity using a neural network as described in 3.3.1. As experimental data is not available freely and in abundance, for the training of this network, the data from the *SGTE* database is used. Whilst it would be possible to apply this network to any element, it is trained only on data from the element iron and its BCC phase. *Länge's* paper also presents the results for iron, thus the results from both networks can be compared well.

As described in Figure 3.6, the network's direct output is the Gibbs energy as a function of the temperature, which is the input to the network. Using the equations described in 3.1.3 the network's parameters can be reused to also calculate the entropy, enthalpy, and heat capacity. *Figure 4.4* shows the basic workflow of the network. The *Python* class, in which the network is implemented, is however designed in such a way that all four functions can be calculated at once.

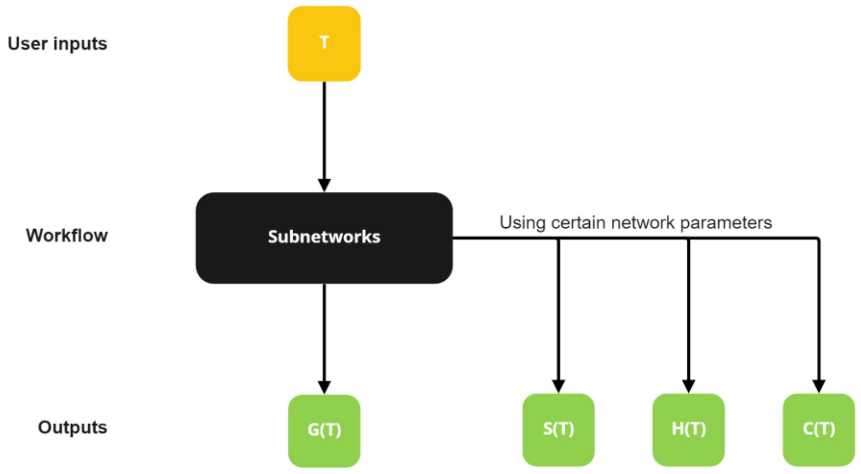


**Figure 4.4:** Basic workflow of the rebuild of *Länge's* network

### 4.2.2 Dataset logic

As shortly described above, the experimental data used in [4] is not easily available, although the *SGTE* database provides similar data in abundance. Therefore, for the training of this network, the *SGTE* data is used by utilizing the data handler described in *4.1.* The *Python* library *PyTorch* provides extensive tools for creating and loading datasets. A *PyTorch* dataset called *LaengeDataset* is created and takes the following arguments for its creation:

- **Temperature range and step size:** the temperature range which the *SGTE* data is loaded for, and the size of steps taken in this range. For further reference, see *4.1.2.*
- **Element:** which element to load the data for

- **Phase:** which phase to load the data for
- **Scaling:** whether to scale the data to the interval from -1 to 1

Using these arguments, the *SGTE* functions are evaluated. The resulting data is then accessible as a combination of a temperature value and the values of the respective functions at this temperature. In case the scaling flag is activated (i.e., the value set to True), the temperature and the Gibbs energy values of the total temperature range are scaled by the respective maximum values to the interval from -1 to 1.

### 4.2.3 Network architecture

For this package, the network presented in *Figure 3.6* is implemented with the information provided in [4]. The paper does not provide information on the number of nodes in the hidden layer of Subnetwork 2, wherefore this number is a variable in the network's definition. It is also not described in the paper how the outputs of the two subnetworks are combined. For this, the assumption is that the outputs are simply added to each other without any weighting factors.

In addition to the network itself, the activation functions are also implemented as *PyTorch* modules so that the parameters in the activation function can be learned during the training process as it was done in [4]. Whilst the softplus activation (equation (*3.37*)) is a standard activation function and therefore available in the *PyTorch* library, its first and second derivatives are not available there. The derivatives are however needed for the calculation of the entropy, enthalpy, and heat capacity. Therefore, the softplus activation is implemented inside the *laenge* package as well as the activation function for subnetwork 1 (equation (*3.36*)), called *ChenSundman* function in the implementation following the use of a paper by *Chen* and *Sundman* [28] in [4] that served as the basis for the use of this activation function. The *ChenSundman* function contains four parameters that are learned throughout the training process (the variables $E_0$, $\theta_E$, $a$ and $b$).

### 4.2.4 Results

The results achieved are not comparable to the results presented in [4]. Especially the approximation of the heat capacity is far off the true values as can be seen in the bottom-left graph in *Figure 4.5*, whereas in [4] it could be approximated well. Also, the prediction of the enthalpy is not accurate (top-left in *Figure 4.5*). The graphs on the right show that both the Gibbs energy and the entropy can be approximated well by the network.

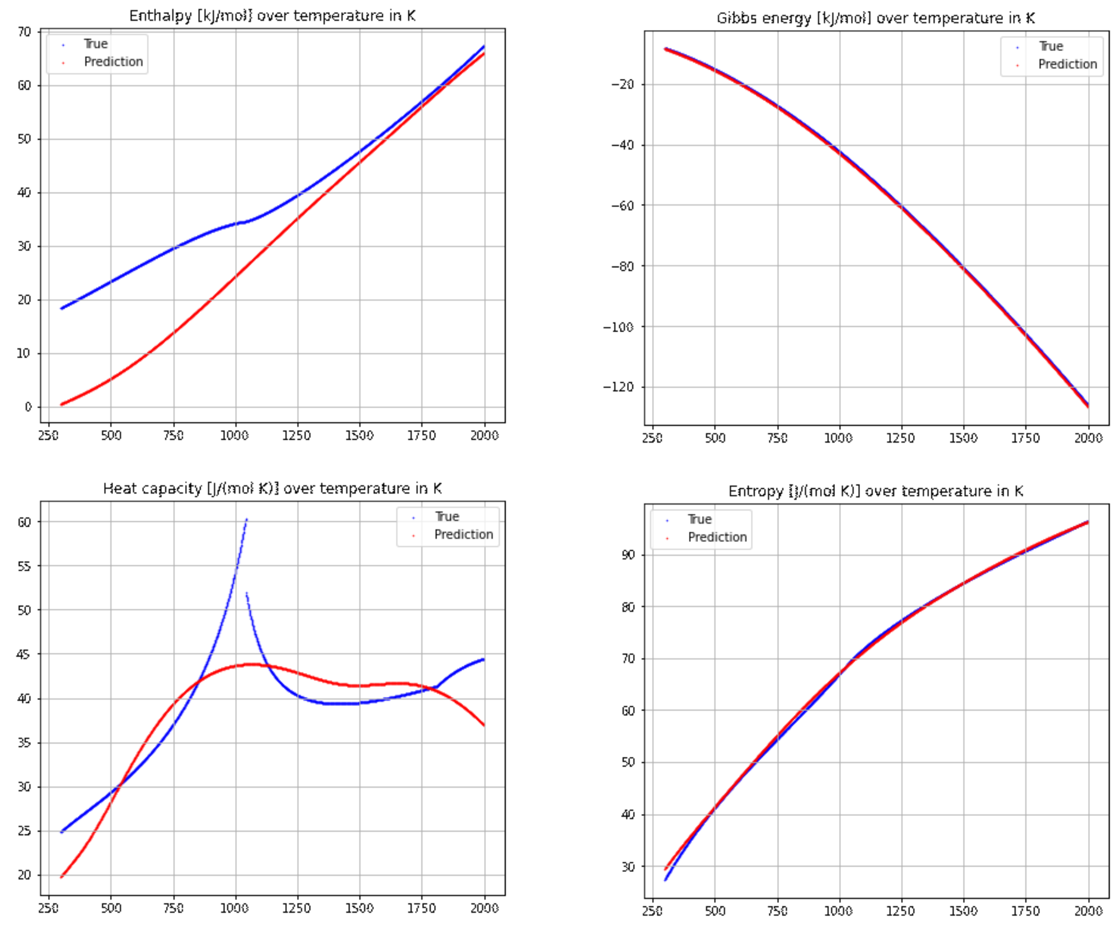


**Figure 4.5:** top-left: approximation of enthalpy, top-right: approximation of Gibbs energy, bottom-left: approximation of heat capacity, bottom-right: approximation of entropy

For the training of the network, a dataset as described above with the following parameters is used:

**Table 4.1:** *LaengeDataset* parameters

| Parameter | Value |
|---|---|
| **Temperature range** | 200K-2000K |
| **Temperature step size** | 2K |
| **Scaling** | False |

The subnetworks are created and initialized using the following hyperparameters:

**Table 4.2:** *LaengeNet* network hyperparameters

| Parameter | Value |
|---|---|
| **Nodes in hidden layer of subnetwork 2** | 32 |
| **Initialization** | Randomly uniform with values between 0.1 and 0.2 |

For the training of the network itself, several adaptations are made to keep the process numerically stable. As the values of Gibbs energy, entropy, enthalpy, and heat capacity vary in magnitude, a scaling is introduced. The true values (to which the network outputs are compared) of the Gibbs energy and the enthalpy are scaled from J to kJ so that the network also outputs values in kJ, which are regarding the magnitude of the same order as the entropy and the heat capacity. Although, the network tends to output small values for entropy and heat capacity, which is why the network's outputs for these capacities are multiplied by 1000.

Also, other than suggested in [4], the variable $\theta_E$ in the loss function is set to a fixed value and therefore not learnable. This results from the fact that $\theta_E$ is an argument of a logarithm in the *ChenSundman* function and is therefore restricted in its possible values. Making this parameter learnable leads in most cases to negative arguments of the logarithm, which is not defined. As a result, the loss cannot be evaluated anymore and therefore learning is not possible. In experiments conducted it proved to be better to fix $\theta_E$ to a certain value rather than restricting the values it can take. For choosing the value, $\theta_E$ is interpreted as the *Einstein temperature*, which is defined as [29]:

$$\theta_E = \frac{h\nu}{k_B} \tag{4.1}$$

There, $h$ is Planck's constant, $k_B$ is Boltzmann's constant and $\nu$ is the atoms' oscillator frequency inside the solid.

Using $\nu = 1.7 \cdot 10^{13}\text{Hz}$ [30] results in a value of $\theta_E = 129.85\text{K}$. The learned parameters of the *ChenSundman* function resulted in:

**Table 4.3:** Results of the learned parameters in the *ChenSundman* function

| Parameter | Value |
|---|---|
| **$E_0$** | -1404.8337 |
| **a** | 0.1356 |
| **b** | -0.0018 |

For the training routine, the loss function from equation (*3.38*) is used. As [4] does not provide information on the weighting values $q$ in the loss functions, a hyperparameter search was conducted to find the optimal values. Alongside the training hyperparameters, the weighting values can be found in *Table 4.4*. The choice of the $q$ values is based on the ratio of the respective losses and $L_G$.

**Table 4.4:** Training routine hyperparameters for the *LaengeNet*

| Parameter | Value |
|---|---|
| **Learning rate** | 0.025 |
| **Number of epochs** | 1000 |
| **Batch size** | 64 |
| $\mathbf{q_G}$ | 1 |
| $\mathbf{q_S}$ | 10 |
| $\mathbf{q_H}$ | 1 |
| $\mathbf{q_C}$ | 10 |

The optimizer used for the training is in accordance with [4] the *Rprop* optimizer.

### 4.2.5 Discussion & Conclusions

#### Discussion

Whilst Gibbs energy and entropy can be approximated well, this is not the case for the enthalpy and the heat capacity. The exact reasons for this are unclear, whereas probably the network's design is the root of the problem. By evaluating the single losses on the respective functions, a back and forth can be noticed, so that the loss on one function improves whilst the loss on another gets worse. Not calculating the entropy, enthalpy, and heat capacity from the network parameters but rather designing the network in such a way that those functions are direct outputs improves the results. Training this network is however only possible when values for all four functions are available at a given temperature. It is possible when using the *SGTE* database as the data source. The results for this network architecture are presented in *4.3*.

In general, the network's architecture and its loss functions lead to numerical instabilities in many different cases. One is the previously described issue with the parameter $\theta_E$ in the activation function of subnetwork 1. Also, the Softplus activation function leads to numerical instabilities in its first and second derivatives. For the first derivative, the problem can be overcome by rewriting the function using the LogSumExp trick [31]. As of equation (*3.37*), the Softplus function is defined as:

$$f_b(s) = \log(e^s + 1)$$

Using the LogSumExp trick, this function can also be written as:

$$f_b(s) = \max(0, s) + \log\left(e^{-|s|+1}\right)$$

The first derivative $f_b'(s)$ therefore is:

$$f_b'(s) = p(s) - \frac{s}{e^{|s|}} \cdot |s| + |s|$$

Where $p(s) = \begin{cases} 1 \ \forall\, s > 0 \\ 0 \ \forall\, s \leq 0 \end{cases}$.

In the second derivative of the Softplus function, small pre-activations $s$ lead to the undefined property $\frac{\infty}{\infty}$, which cannot be handled by *PyTorch.* Therefore, the learning algorithm crashes once this case occurs. It can although be overcome by replacing those values with 0.

Also, the network initialization can lead to numerical instabilities. Specifically, all parameters must be initialized with either all positive or all negative values. A very common initialization with values drawn from a standard normal distribution can therefore not be applied. Learning rates higher than 0.01 also lead to instabilities in some cases.

**Conclusions**

It can be concluded that the network architecture and the choice of loss functions are very prone to numerical instabilities. Also, the results when trained on the *SGTE* data are poor. The numerical instabilities can be overcome by introducing certain restrictions which make training possible. Once the network is trained, the training algorithm is not needed anymore, wherefore the difficult training routine could be accepted. Although, this is not possible for the poor results. Therefore, an approach is made in *4.3* to design a network that can be trained without numerical instabilities and that delivers better results.

## 4.3 Function approximation for unary systems – thermonet

### 4.3.1 Goal and workflow

As chapter *4.2* concludes that the network presented in [4] cannot be easily rebuilt, a different approach to achieve the same approximations is presented in this chapter. The main goals of this network are to avoid numerical instabilities in the training process as well as to achieve better and more accurate results. To do so, the approach of deriving the entropy, enthalpy, and heat capacity from the network parameters using partial differential equations is abandoned. Instead, all four capacities are direct outputs of the network as can be seen in the basic workflow in *Figure 4.6.*

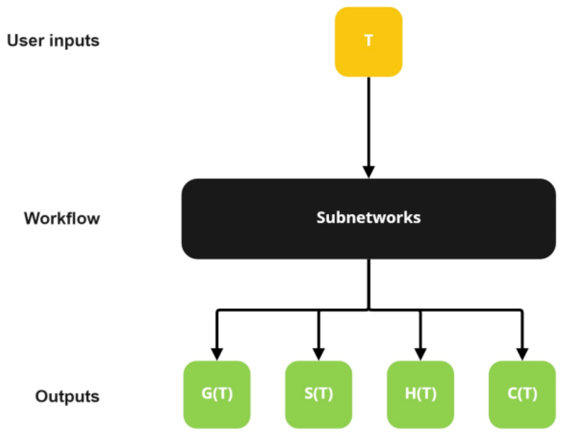


**Figure 4.6:** Basic workflow of the *ThermoNet*

### 4.3.2 Dataset logic

The basic idea behind the dataset for the *ThermoNet* is very similar to the one for the *LaengeNet*, although it differs from it in some points. The main difference is that the *LaengeDataset* presented in *4.2.2* can only contain data from one element. To allow more flexibility, the dataset for the *ThermoNet*, called *ThermoDatasetNew,* can load and contain data for multiple elements at once. If only one element's data is loaded, the phases to be loaded can be specified, otherwise, all phases of all the desired elements must be loaded. The dataset takes the following arguments for its creation:

- **Temperature range and step size:** the temperature range which the *SGTE* data is loaded for, and the size of steps taken in this range. For further reference, see *4.1.2.*
- **Elements:** which elements to load the data for
- **Phase selection:** a flag called *inp_phases* that defines whether all phases or just a selection are loaded in case only one element's data is contained in the dataset.

Using these arguments, the *SGTE* functions are evaluated. The resulting data is then accessible as a combination of a temperature value and the values of the respective functions at this temperature. As opposed to the dataset for the *LaengeNet*, a scaling of the values in the dataset is not possible.

### 4.3.3 Network architecture

Unlike the *LaengeNet* (see *Figure 3.6*), where the only network output is the Gibbs energy and all the other capacities are derived from the network parameters, *ThermoNet* has four outputs, one for each of Gibbs energy, entropy, enthalpy, and heat capacity. This architecture is chosen as the design of the *LaengeNet* with the derivation of the entropy, enthalpy, and heat capacity using derivatives of the network parameters was identified as a possible reason for its poor results. A schematic of the architecture can be seen in *Figure 4.7*. All layers are fully connected but for improved clarity, no connections between the nodes are shown. Every hidden layer and the input layer use the softplus function as their activation function, whereas the output does not use an activation function.

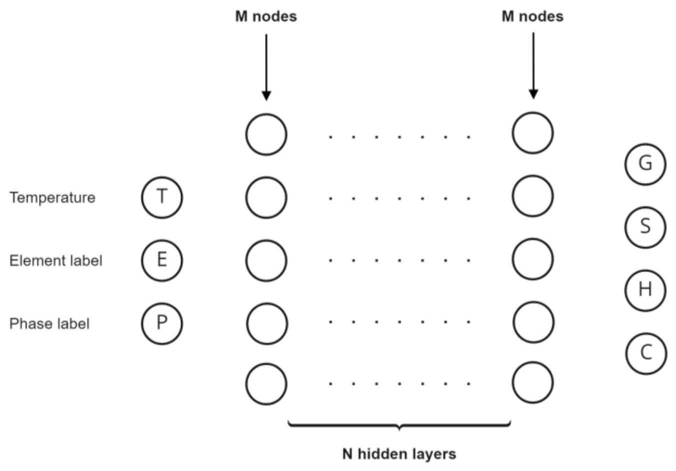


**Figure 4.7:** Network architecture of the *ThermoNet*

The network is made up of $N$ hidden layers with $M$ nodes each. Those two values are hyperparameters that are optimized in the training process. The numbers of nodes in the input and output layers are fixed to 3 respectively 4, as the network receives a temperature value as well as a numeric element and phase labels as input and returns its predictions for the four functions. The element and phase labels are introduced to make the network more flexible in terms of what can be predicted. With this implementation, it is possible to train the network on the data of multiple elements and phases simultaneously, which is not possible with the *LaengeNet*.

## 4.3.4 Results

### Training on pure iron data

The results of the *ThermoNet* proved to be better than the ones achieved with the rebuilding of the *LaengeNet.* Not only the Gibbs energy and the entropy as in the case of the *LaengeNet* but also the enthalpy and the heat capacity can be approximated well for the BCC phase (*Figure 4.8*) and the FCC phase (*Figure 4.9*) of iron when trained only on the iron data.

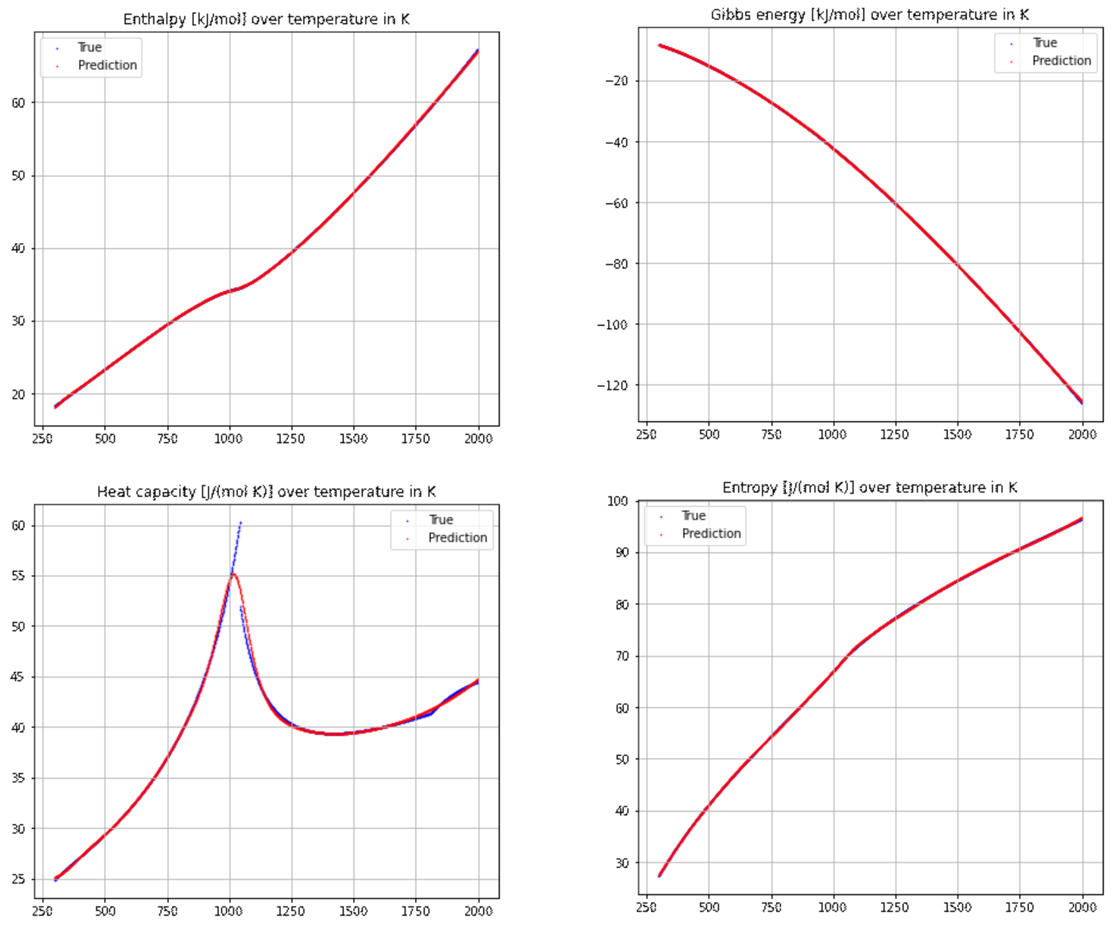


**Figure 4.8:** approximation of the properties for the BCC phase of iron; top-left: approximation of enthalpy, top-right: approximation of Gibbs energy, bottom-left: approximation of heat capacity, bottom-right: approximation of entropy

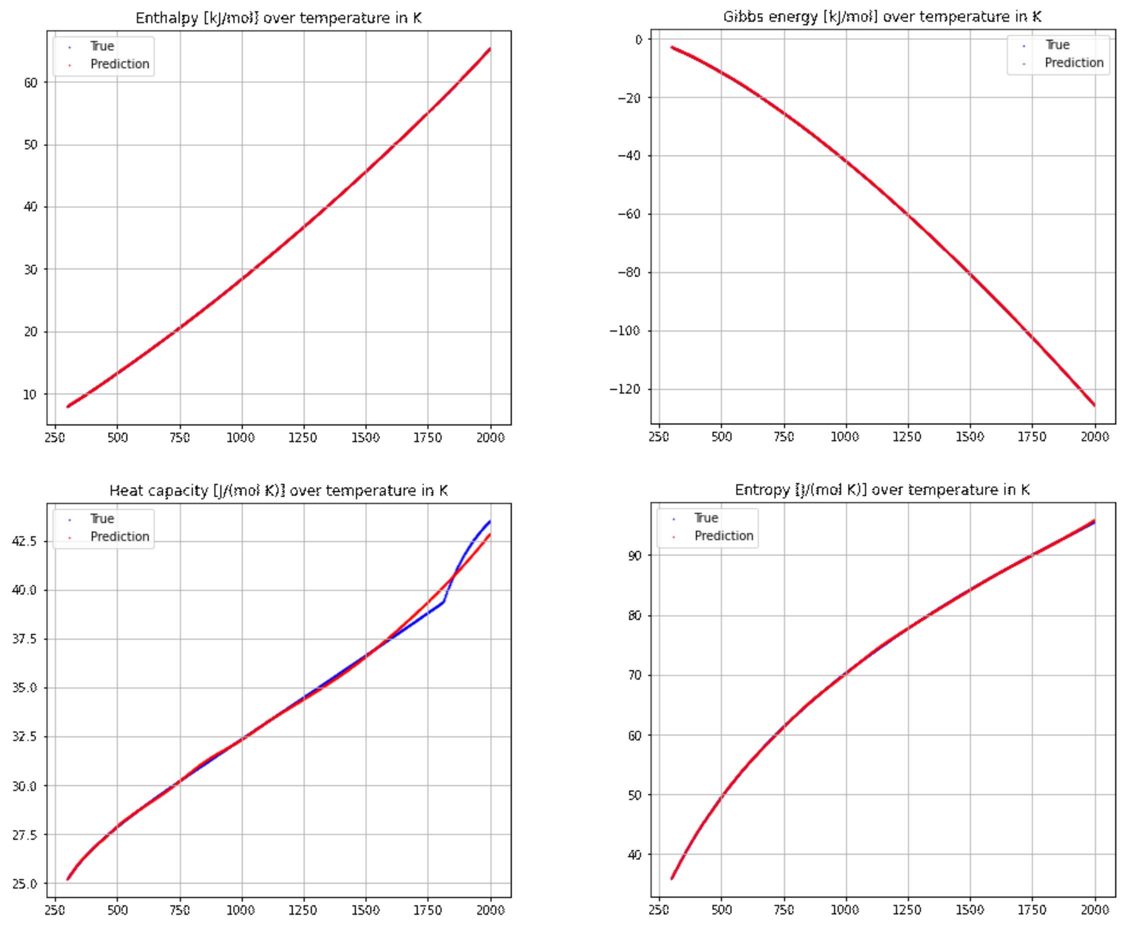


**Figure 4.9:** approximation of the properties for the FCC phase of iron; top-left: approximation of enthalpy, top-right: approximation of Gibbs energy, bottom-left: approximation of heat capacity, bottom-right: approximation of entropy

For the training of the network, a dataset as described above with the following parameters is used:

**Table 4.5:** *ThermoDatasetNew* parameters for training on pure iron data

| Parameter | Value |
|---|---|
| **Temperature range** | 200K-2000K |
| **Temperature step size** | 1K |
| **Elements** | Fe |
| **Phase selection** | None (all phases are selected) |

The network is created and initialized using the following hyperparameters:

**Table 4.6:** *ThermoNet* network hyperparameters

| Parameter | Value |
|---|---|
| **M** | 128 |
| **N** | 8 |
| **Initialization** | Xavier normal |

The training is conducted by loading temperature values and the respective element and phase labels from the dataset and feeding it into the network. The output is then compared to the true values using the mean squared error loss. Also in [4], the mean squared error loss is used to evaluate the network's predictions, whereas one mean squared error loss is used for each function. The resulting total loss is obtained by weighing and adding up the losses (equation (*3.38*)). Calculating the mean squared error on all the four functions at once can also be seen as calculating four single errors. Those errors are however not weighted by fixed weighting parameters and contribute evenly to the total loss.

At the beginning of the training, high errors occur. Choosing a high enough learning rate can reduce the error after a few epochs by around $10^2$. At this point, the predictions are still far off the desired outputs, whereas the learning rate becomes too big to furtherly improve and minimize the error. Instead, the error starts to oscillate. The oscillation can be reduced by reducing the learning rate, which as a result also minimizes the error. Therefore, a learning rate scheduler is used. A learning rate scheduler allows starting with a high learning rate which is then reduced after every epoch by a defined value. In this case, an exponential learning rate scheduler, which decays the learning rate by the factor $\gamma$ every epoch, is used.

As the learning rate is part of the optimizer, the learning rate scheduler and the optimizer are closely related. To find the best optimizer for this case, all optimizers available in the *PyTorch* library were tested, whereas some of them worked well and others did not at all. The best results are obtained by using the RMSProp optimizer.

The training is conducted using the following hyperparameters:

**Table 4.7:** Training routine hyperparameters for the *ThermoNet*

| Parameter | Value |
|---|---|
| **Initial learning rate** | 0.005 |
| **Number of epochs** | 2500 |
| **Batch size** | 1028 |
| **$\gamma$** | 0.995 |

## Training on iron and carbon data

The network is designed in such a way that it can be used for the approximation of the functions for multiple elements at once (*Figure 4.7*). To test this flexibility, the network is trained not only on the data for pure iron as described above but additionally also on the data for carbon. As in this case, also the element label plays a role in the prediction, it can be assumed that the training is harder and that the results will be worse compared to the results presented above. This assumption proves to be true, as *Figure 4.10* shows for the approximation of the functions for the BCC phase of iron.

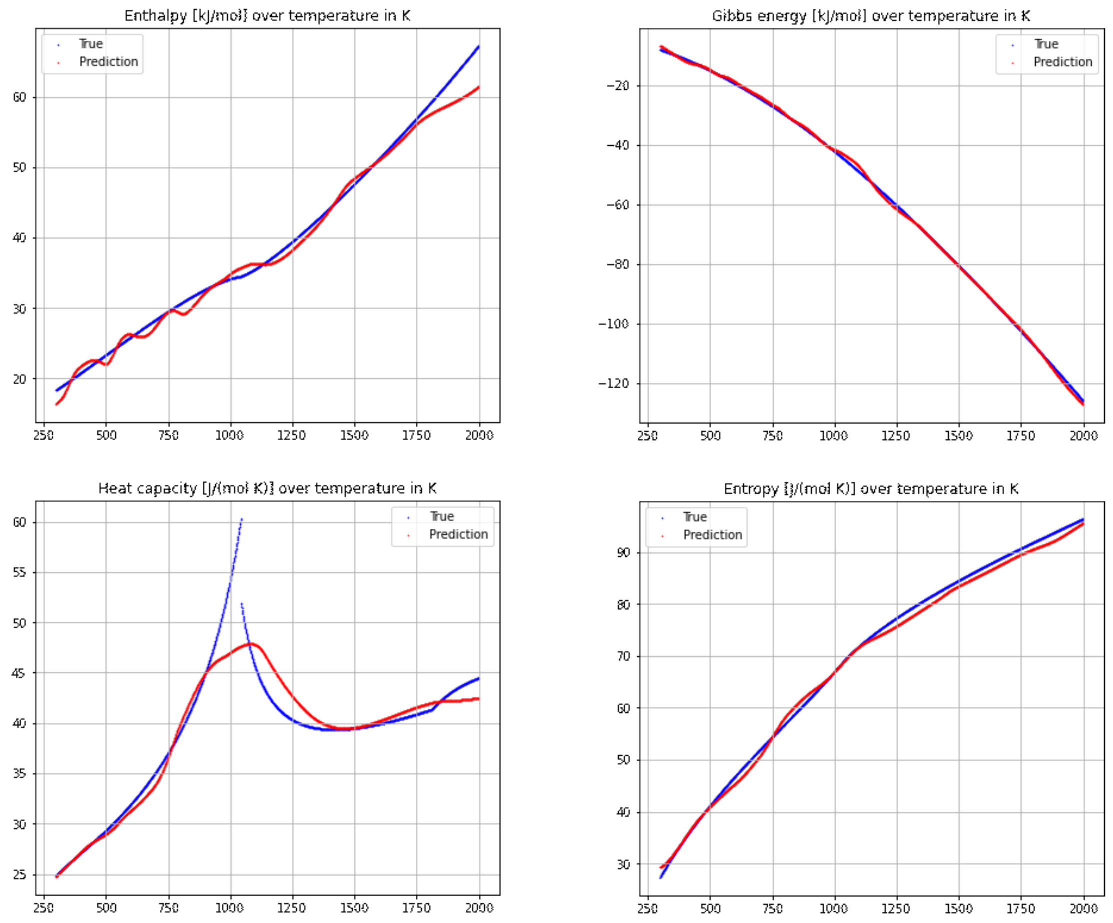


**Figure 4.10:** approximation of the properties for the BCC phase of iron when trained on iron and carbon data; top-left: approximation of enthalpy, top-right: approximation of Gibbs energy, bottom-left: approximation of heat capacity, bottom-right: approximation of entropy

These results, especially for the entropy and the enthalpy, are significantly worse than when the network is trained only on the iron data. On the other hand, the results are still better than the ones obtained from *LaengeNet*. As the network is also trained on data for carbon, it can also make predictions on the functions for carbon. The results are similar for all phases of carbon, whereas the graphite phase presented in *Figure 4.11* yields the most accurate results.

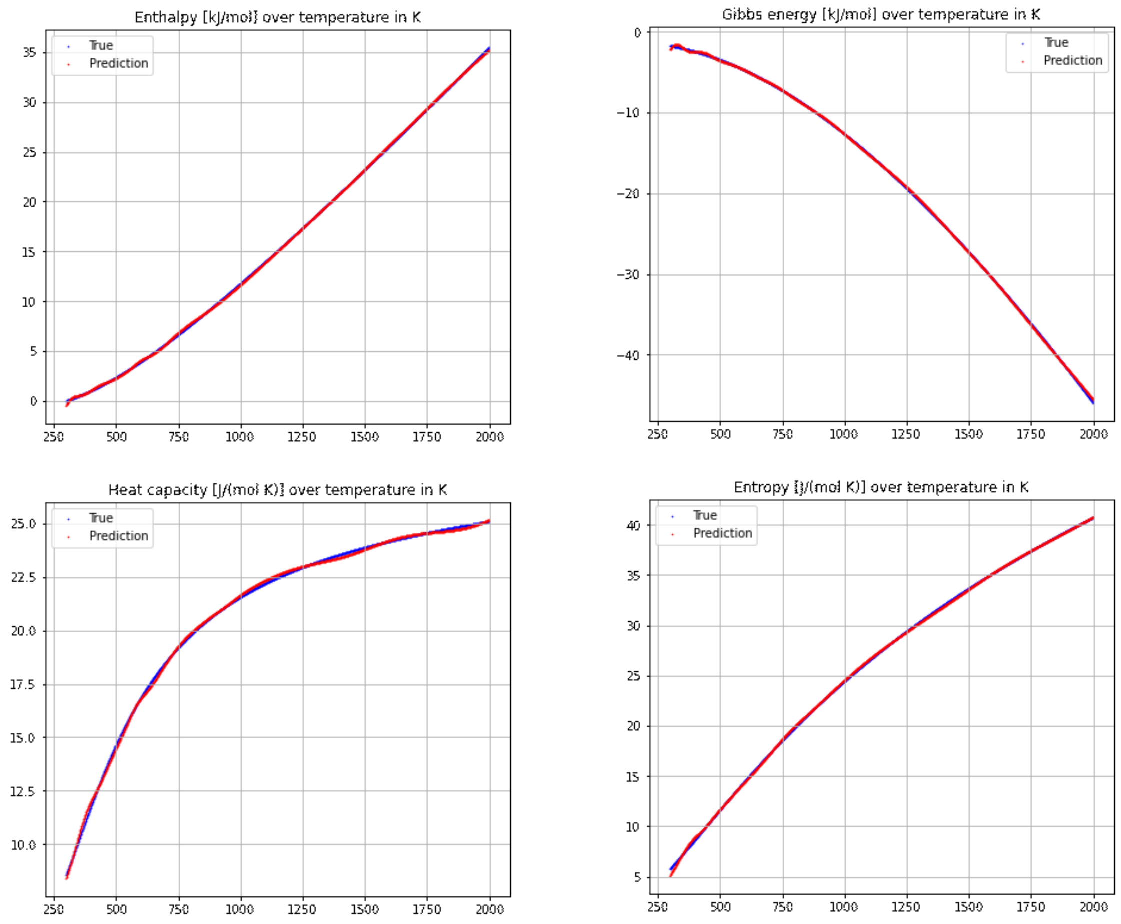


**Figure 4.11:** approximation of the properties for the graphite phase of carbon when trained on iron and carbon data; top-left: approximation of enthalpy, top-right: approximation of Gibbs energy, bottom-left: approximation of heat capacity, bottom-right: approximation of entropy

For the training of the network, a dataset with the following parameters is used:

**Table 4.8:** *ThermoDatasetNew* parameters for training on iron and carbon data

| Parameter | Value |
|---|---|
| **Temperature range** | 200K-2000K |
| **Temperature step size** | 1K |
| **Elements** | Fe, C |
| **Phase selection** | None (all phases are selected) |

All the parameters presented in *Table 4.6* and *Table 4.7* are reused for the training of this network. Also, the learning rate scheduler, optimizer and loss function are the same as described above.

### 4.3.5 Discussion & Conclusions

#### Discussion

As *Figure 4.8-Figure 4.11* show, the results obtained with this network are much better than the results with the *LaengeNet* (*Figure 4.5*). With neural networks, it is often hard to determine what the cause for certain behaviors is. It is although evident, that the changed network architecture plays a major role in it. In this architecture, all four network outputs depend on the network parameters in the same way, whereas in the case of the *LaengeNet*, they depend on the first and second derivatives of the activation functions as well as the pre-activation. Optimizing such an architecture is harder.

Also, thanks to the architecture, the training is possible without encountering numerical instabilities. Compared to *LaengeNet*, this is possible because of the choice of activation functions. Whilst the *ChenSundman* function introduces physics into the learning process, it also brings numerical issues with it. Especially the learnable parameters in the activation function, as well as the first derivatives of both the ChenSundman and the softplus functions, are roots of instabilities.

Even though this network performs better than the *LaengeNet*, it comes with the drawback that training is only possible when data for the Gibbs energy, entropy, enthalpy, and heat capacity is available for the same temperature values. Therefore, in [4] the temperature input does not need to be the same for the four functions. Of course, it would be possible to design the network in such a way that it can receive different input temperatures for all the functions in question, although for the sake of simplicity this was not done in this case.

The big advantage the *LaengeNet* has against the *ThermoNet* is that the *LaengeNet* can also be interpreted as four standalone networks. Therefore, it is possible to train the network only on one property and receive the others from it. This is advantageous because fewer data and therefore fewer measurements are needed and because it might be difficult to measure some of the properties.

It is also possible to train this network on the data of multiple elements at once. As *Figure 4.10* shows, this leads to worse results than a network trained on the data of only one element (*Figure 4.8*). As *Figure 4.10* and also *Figure 4.11* show, in this case, the approximated functions tend to oscillate around the actual function values. To reduce these oscillations, introducing a term to the loss function that takes the approximations' curvature into account could improve the results.

#### Conclusion

When directly comparing the networks, the *ThermoNet* is the better option both in terms of accuracy as well as ease of training. This comes with the drawback that data must be available for all four functions at the same temperature values, which is not necessarily required for the *LaengeNet*. For the practical use-case of developing a material model of an element where there are measurements available for only one property, say the heat capacity, the *LaengeNet* is better suited. Although, a way to overcome the numerical instabilities needs to be found. In case the training of the network is conducted on data for all four properties, the *ThermoNet* or a similar architecture is better suited.

As the comparison of the results between the network trained only on the pure iron data and the network trained on iron as well as carbon data shows, it is better to train single networks for every element because it yields better accuracy. Even though accurate approximations of the Gibbs energy, entropy, enthalpy, and heat capacity could be achieved with this network, its only use is for demonstration purposes. It is trained on the *SGTE* data and therefore only a different representation of this dataset. Every output the network can provide is also directly accessible from the *SGTE* database, either by solving the equations or using software like *FactSage* or even the *Python* library presented in *4.1.* This although does not mean that the network is useless. Rather than that, it can be used to derive a material model from experimental data. As described above, a choice for the architecture must be made depending on which data is available.

## 4.4 Equilibrium prediction for binary systems – binary predictor

### 4.4.1 Goal and workflow

The goal of the algorithms implemented in this package is to calculate binary phase equilibria given analytical Gibbs energy functions as a function of the composition $x$ and the temperature $T$ at a given pressure $p$. Using equations (*3.29.1*) and (*3.29.2*), the equilibria compositions can be found (if an equilibrium exists for this temperature). Therefore, the goal is to find composition values for either phase that fulfill those equations. The problem that arises although is that it cannot be told a priori, how many solutions (i.e., single equilibria) to this equation system exist. To implement a neural network, it is however necessary to define a fixed number of output nodes. Three different approaches were taken into consideration:

1. Restricting the possible systems to such that have only one equilibrium at each temperature. In this case, the number of output nodes could be set to 2, one node for the equilibrium composition of either phase. The advantage of this approach is that the accuracy of the results will likely be very good, the disadvantage is that the predictor is applicable to only a very small number of systems.
2. Another approach is to predetermine the number of single equilibria that will or can occur in a system given the Gibbs energy functions. This could either be done by logical rules or by training a classification neural network on this task. Given the number of predetermined equilibria, a network could be selected that has the exact number of output nodes needed. Multiple networks would be necessary to account for every number of single equilibria. Also, this method likely has very good accuracy but comes with many drawbacks: an additional pre-processing step would have to be introduced and in the case of a classification neural network, for this aim, an additional network

would have to be trained. Also, a balanced amount of function pairs, so that each number of equilibria occurs the same time in the training and test sets, must be found. In addition, in all cases with more than one single equilibrium, it is not guaranteed that the network finds all equilibria. It is possible, that for the example of two single equilibria in the system, both outputs are for the same equilibrium.

3. The third approach is the most general one as it does not restrict the possible systems and does not need sophisticated pre-processing. Both equations in equation (*3.29*) are formulated in such a way, that the respective left sides of the equations depend on the equilibrium composition $x^{\alpha}$ of phase $\alpha$ and the respective right sides on the equilibrium composition $x^{\beta}$ of phase $\beta$. It is therefore possible, to find functional relationships $x^{\alpha} = x^{\alpha}(x^{\beta})$ for either equation so that the equations are fulfilled. The intersection points of the resulting two equations are the solution to the equilibrium problem. To find the functional relationships, two neural networks (one for each function) are used. The advantage of this approach is its generality, which comes with the risk of greater inaccuracy than the two approaches presented above. Additionally, the network does not directly output the equilibria compositions. Moreover, they must be determined in a post-processing step.

As the third approach allows the best generality, it is the approach chosen to be implemented. The risk of loss in accuracy compared to the other two approaches is accepted.

The basic workflow of the algorithm can be seen in *Figure 4.12.* As described above, two networks are used in this approach. One finds $x^{\alpha} = x^{\alpha}(x^{\beta})$ so that the chemical potential is equal for both phases (equation (*3.29.1*)), whilst the other finds a similar functional relationship so that equation (*3.29.2*) is fulfilled. Both networks receive as input the values of the Gibbs energy functions (in the figure below called $G'$ and $G''$, in equation (*3.29*) called $G^{\alpha}$ and $G^{\beta}$) evaluated at a fixed number of composition values as well as at a constant temperature. The predicted functional relationships are then passed to a post-processing algorithm which finds the intersection points between the two network outputs so that as a result, the composition values at the equilibria (in the figure below called $x'\_eq$ and $x''\_eq$) can be found. For the predicted functional relationship, for $x^{\beta}$ the same x-values that are used to evaluate the Gibbs energy functions are used. Therefore, $x^{\alpha} = x^{\alpha}(x^{\beta})$ is not a continuous function but rather a function evaluated at certain points. As a result, an algorithm to find the intersection points and to avoid duplicated equilibria compositions is necessary.

When evaluated at different temperatures (or also different pressures), phase diagrams for the given system can be drawn. To do so, the Gibbs energy functions need to be evaluated at different temperatures (or pressures) and passed to the algorithm. Depending on the temperature granularity, this will lead to longer or shorter execution times.

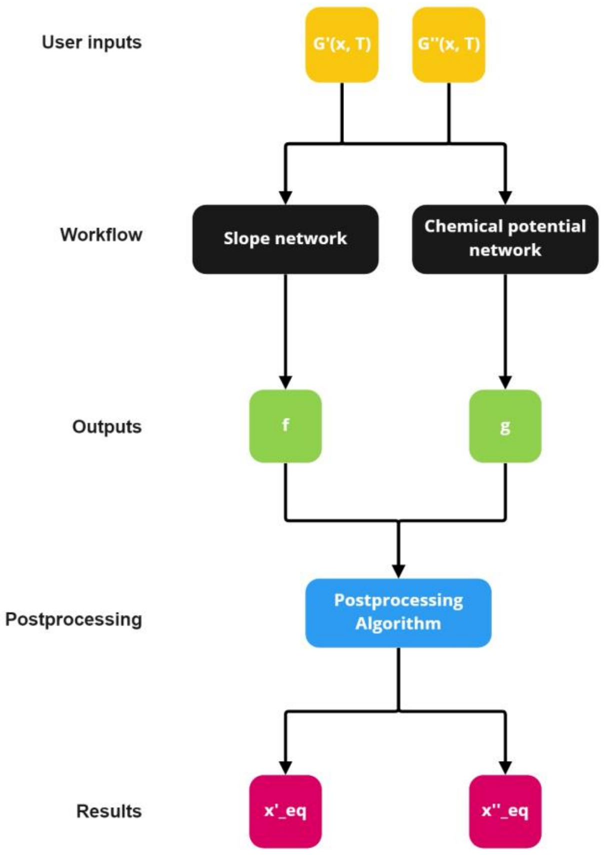


**Figure 4.12:** Basic workflow of the binary predictor

### 4.4.2 Dataset logic

The dataset for the training is compiled in a *Python* class called *FunctionPairDataset (FPD)* which builds on another *Python* class called *FunctionDataset*. In the *FPD*, functions can be generated following the rules presented in [12]. There, examples of possible Gibbs energy functions depending on certain parameters and rules for which values these parameters can take, are published. Following the presented rules, a provided number of pairs of functions is generated using random coefficients. The arguments can be specified for the initialization of the dataset:

- **Number of functions:** the number of function pairs to be generated
- **Filename:** a filename and -path where the parameters used to generate the functions can be stored at
- **X step size:** The step size taken in the x-range from 0 to 1. At every resulting x value, the functions are evaluated.
- **Overwrite:** If necessary, the file where the parameters are stored can be overwritten if the filename already exists. In case this flag is set to False and the file already exists, the functions are reloaded using the parameters from the file.

Every function needs a temperature at which it is evaluated as well as melting temperatures of either phase as parameters. Those temperatures are drawn randomly in the range from $0 - 3000K$. Also, entropy values can be part of the equations. In case entropy is included, the values are also drawn randomly in the range from 0 to 10 $\frac{J}{mol\,K}$. The dataset compiles the specified number of functions and returns the evaluated values divided by the global absolute maximum of both functions, the parameters used for the generation, and the absolute maximum value of both phases. The functions are divided by this maximum value so that the values are scaled to the range from -1 to 1. This is because those values are passed to the neural network as input and by doing so, it can be guaranteed that the network always receives inputs of the same magnitude.

For the testing of the network, the same *Python* classes can be reused. It is also possible to construct the functions outside the dataset class using the respective functions that implement the core ideas of [12]. *Figure 4.13* shows two exemplary functions generated using the Mager-Lukas-Petzow rules for different values of $A_{\text{diff}}$ and $A_{\text{sum}}$, which are parameters in the equations. Both functions are evaluated for a temperature of $1000K$ and melting temperatures of $900K$ for one and $1100K$ for the other component.

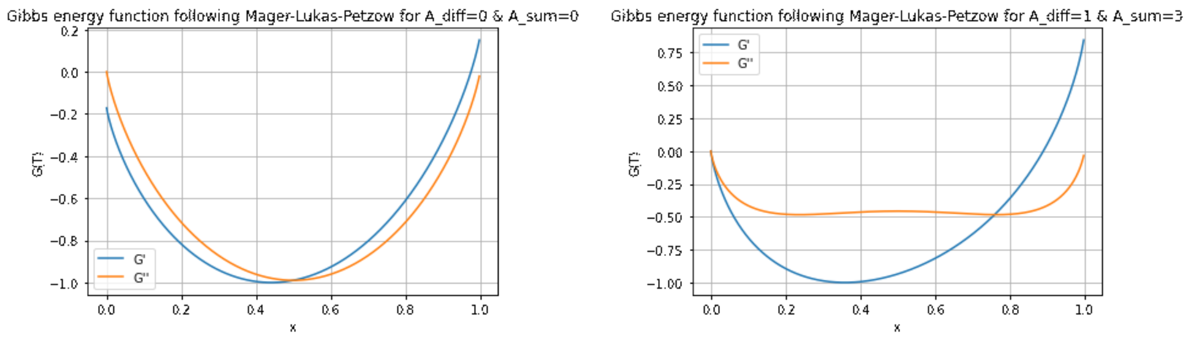


**Figure 4.13:** Examples for function pairs compiled using the Mager-Lukas-Petzow rules; left: $A_{\text{diff}} = 0, A_{\text{sum}} = 0$, right: $A_{\text{diff}} = 1, A_{\text{sum}} = 3$

### 4.4.3 Network architecture

As shown in *Figure 4.12,* two networks are used in the workflow. The network class (called *TangentNet*) was therefore implemented flexibly so that the same class can be used for either network. *Figure 4.14* shows the network's architecture. A number $N$ of hidden layers with $M$ nodes each can be chosen. The number of nodes in the input layer is exactly two times the number of nodes in the output layer. This is because the network receives as input the Gibbs energy functions evaluated at a certain number of points. The output $x^{\alpha} = x^{\alpha}(x^{\beta})$ is a function the is evaluated at the exact same points, therefore the number of output values is only half the number of input values.

All layers except the output layer are activated by the ReLU function. Because the composition can only take values between 0 and 1, the output layer is activated by the Sigmoid function which maps all values to this range.

It should be noted that the network can get big depending on the granularity chosen. The results are more accurate the more points the functions are evaluated at. This comes with the drawback that the number of parameters in the network increases. Therefore, the execution time of the algorithm also increases. A good compromise between accuracy and calculation speed therefore must be found.

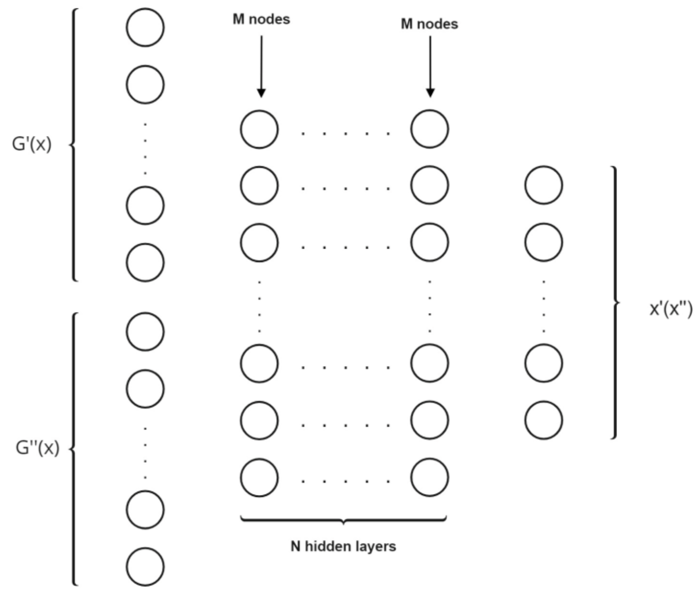


**Figure 4.14:** Network architecture of the binary predictor (' refers to $^{\alpha}$ and '' refers to $^{\beta}$ in previous equations)

### 4.4.4 Description of the algorithm

#### Requirements for the algorithm to work & pre-processing

For the algorithm to work, two functions and the respective first derivatives need to be known as analytical functions. In *Python,* they must be implemented as a callable (e.g., a function or a lambda function) and take as input a composition and a temperature value. Together with a temperature range, those functions are passed to the function *get_phase_diagram* which is part of the *BinaryPredictor* class. Also, a threshold value, which determines the maximum difference between the slope of a predicted common tangent and the slopes of either curve and a maximum threshold value (if for the given threshold no tangents are found, the threshold is increased step-wise up to the maximum threshold) can optionally be specified.

By calling the *get_phase_diagram* function, the pre-processing is initiated. In the pre-processing, at first, a check is made on whether one phase is stable over the whole composition range. In this case, the values of the Gibbs energy function of this phase are lower than the values of the Gibbs energy function of the other phase for all composition values. *Figure 4.15* shows an example of this case. There, the solid phase is stable everywhere because the solid curve's values are always smaller than the liquid curve's. As there cannot be any common tangents, in this case, the algorithm is aborted for this temperature and will be continued with the next temperature in the range.

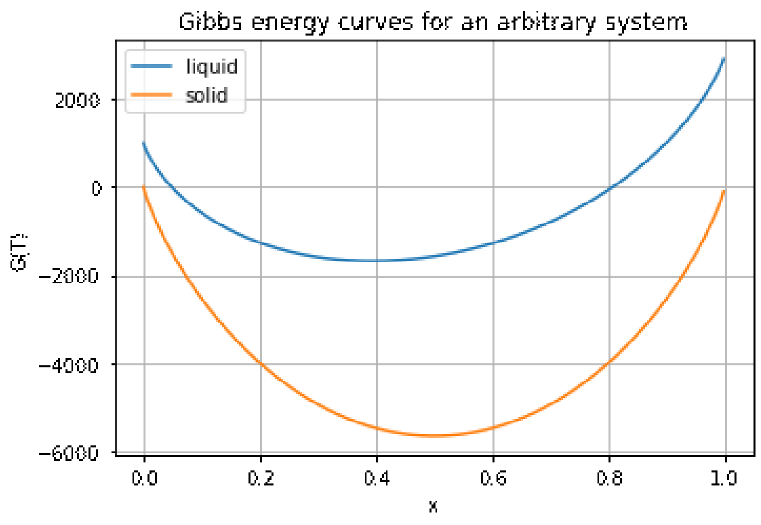


**Figure 4.15:** Gibbs energy curves for an arbitrary system, where the solid phase is stable everywhere

For temperatures where there are common tangents (i.e., equilibria between the two phases), the absolute maximum value of both phases is determined, so that the functions can be scaled by this value to the range from -1 to 1. With this, it is made sure that the networks always receive inputs of the same magnitude. Even though this is already done when creating the training set, the functions may not be scaled in a testing application, wherefore it must be done again. *Figure 4.16* shows a pair of scaled Gibbs energy curves where there will be one common tangent and therefore one phase equilibrium.

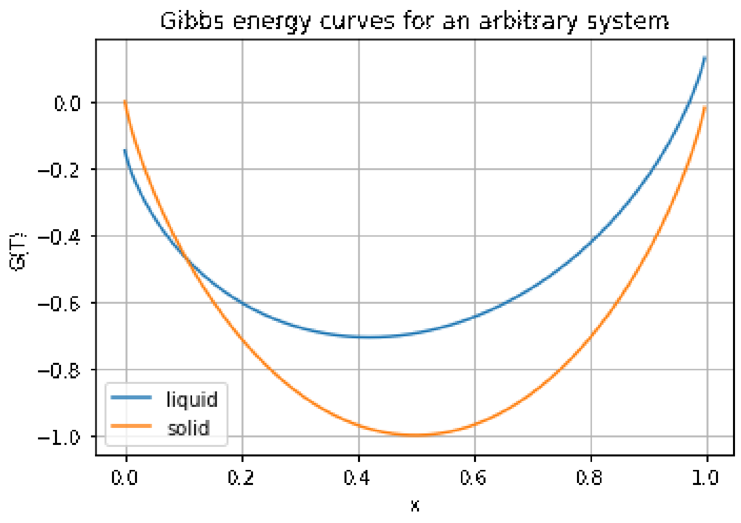


**Figure 4.16:** Scaled Gibbs energy curves

## Neural network predictions

After the pre-processing step, the Gibbs energy curves evaluated at predefined composition values and a temperature are stacked together in one vector and passed to the networks. The networks make predictions for $x^{\alpha} = x^{\alpha}(x^{\beta})$ so that equations (*3.29*) are fulfilled. As the output of both networks are functions of $x^{\beta}$, they can be plotted in the same graph (*Figure 4.17*). The outputs are then passed to the post-processing to final find the equilibria.

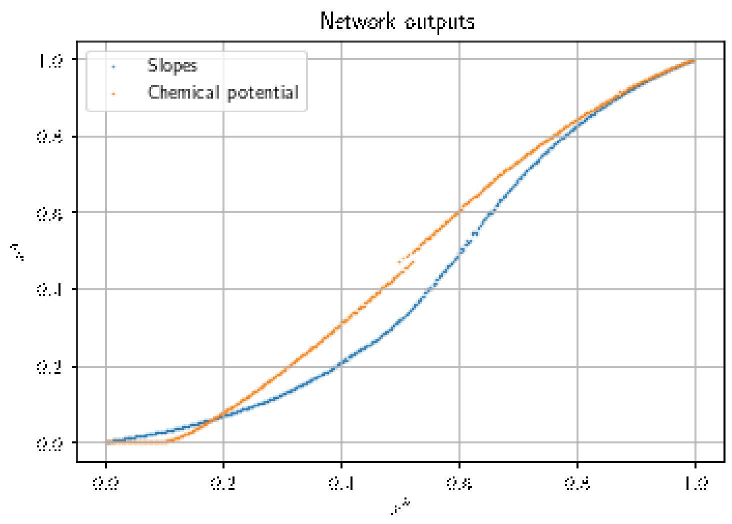


**Figure 4.17:** Outputs of both networks

## Post-processing

Both equations in equation (*3.29*) are fulfilled when there is an intersection of the two curves shown in *Figure 4.17.* Therefore, the intersections need to be found. This is the same as finding the roots of the difference between the two output functions (*Figure 4.18*).

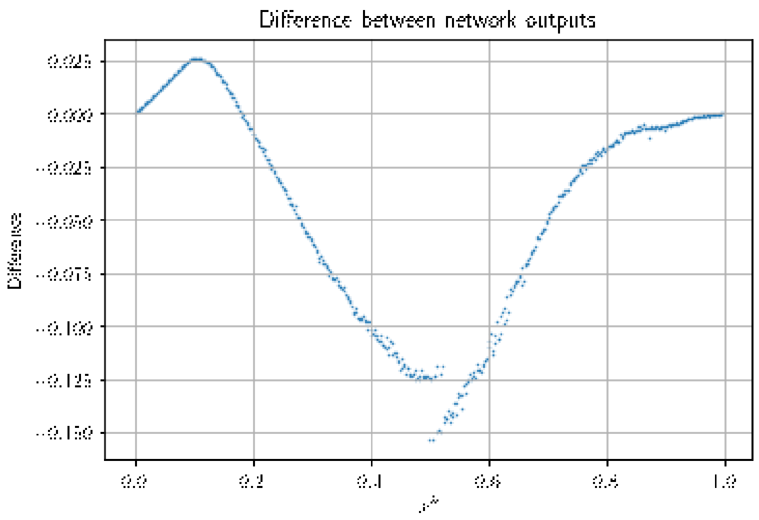


**Figure 4.18:** Difference between the network outputs to find the intersection points

As the network outputs are approximations, the intersections will in general not be exactly where the equilibrium is, i.e., where the equations (*3.29*) are fulfilled. Therefore, all value differences that are smaller than a predefined threshold (chosen to be 0.1) are considered. For all these values, the corresponding $x^{\alpha}$ values will be chosen using the network outputs.

From there, the slopes of the resulting possible tangents are calculated and compared to the slopes of either curve. All tangent candidates, for which the difference between the slope of the tangent and the slopes of the curves is smaller than a predefined threshold, are kept, otherwise, they are discarded. In case no tangent is found, the threshold is increased by 0.1 up to a user-defined maximum threshold.

In the next step, duplicates are sorted out, because it is not unlikely that composition values are kept that form the same tangent as other pairs of composition values. To do so, all the composition values are rounded off to the first decimal point and then paired with the composition values. In the last step, the pairs with the smallest squared difference of slopes to the curves are kept. The example below will demonstrate the process.

---

**Example:** Assume there are two tangents to be found. After considering the difference between the network outputs and the deviations from the curves' slopes, 7 tangents are suggested, whereas there are multiple "duplicates". The duplicates are sorted out using the following process. As a measurement variable, the square root of the sum of the squared differences between the slope of the tangent and either curve is used.

The following composition values are considered tangent candidates:

**Table 4.9:** Tangent candidates

| $x^{\alpha}$ | $x^{\beta}$ | Slope difference |
|---|---|---|
| 0.1089 | 0.2475 | 0.324 |
| 0.1023 | 0.2489 | 0.298 |
| 0.0987 | 0.2503 | 0.315 |
| 0.6498 | 0.9132 | 0.212 |
| 0.6543 | 0.9089 | 0.204 |
| 0.6517 | 0.9099 | 0.225 |
| 0.6487 | 0.9112 | 0.218 |

Then, the composition values are rounded off to the first decimal point and clustered based on equal values. For example, the first line in *Table 4.9* will be rounded off to the pair 0.1 and 0.2, the same as the values in the second line. Therefore, they are clustered together and the one with the smallest slope difference is kept. This is done for all clusters.

**Table 4.10:** Rounded composition values with the minimum slope differences

| $x^{\alpha}$ | $x^{\beta}$ | Smallest Slope difference |
|---|---|---|
| 0.1 | 0.2 | 0.298 |
| 0.0 | 0.2 | 0.315 |
| 0.6 | 0.9 | 0.204 |

As one can see, lines 1 and 2 in *Table 4.10* belong to the same tangent but are not clustered together because of the nature of this process. To avoid this, in the last step, the composition value pairs are interpreted as vectors and the distances between the points are computed. In case the distance is smaller than a predefined threshold (chosen to be 0.2), the value pairs are considered to belong to the same tangent and again, the one with the smallest slope difference is kept. It would be possible to calculate the distance between the composition values right from the beginning, it although proved to be much faster by applying the above-described pre-sorting. All composition values that are kept after this step are considered the results and returned.

For the function pair in *Figure 4.16* (with the network outputs in *Figure 4.17* and *Figure 4.18*), the correct tangent is found (*Figure 4.19*). The x-value of the red dot on the liquid (i.e., blue) curve is the equilibrium composition of the liquid phase, whereas the x-value of the red dot on the solid curve is the equilibrium composition of the solid phase.

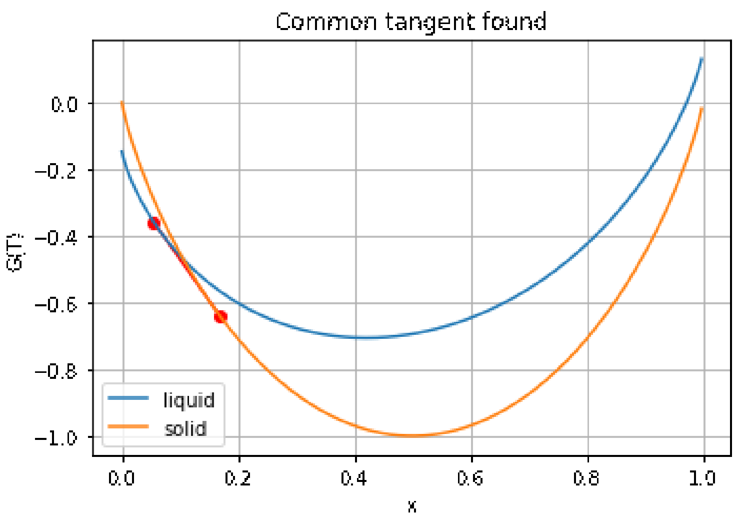


**Figure 4.19:** Common tangent found for the example system

## 4.4.5 Results

For testing the network, two different approaches are chosen. On the one hand, the phase diagram of the Au-Ag system based on the Gibbs energy functions for this system presented in [32] is calculated and compared to the actual phase diagram taken from the thermodynamic software *ThermoCalc.* On the other hand, [12] is used to generate arbitrary functions the same as in the generation of the training set although with different coefficients. Whereas for the Au-Ag system a deviation from the real values can be calculated, this cannot be done for the arbitrary functions as there the ground truth is not known.

For all approaches, the same networks and therefore the same training routine is used. As described in *4.2.2,* the training dataset is generated following the rules from [12] using random coefficients. The dataset is generated using the following arguments:

**Table 4.11:** *FunctionPairDataset* parameters for training

| Parameter | Value |
|---|---|
| **Number of composition values where functions are evaluated** | 500 |
| **Number of function pairs** | 100.000 |
| **Temperature range** | 0-3000K |
| **Entropy range** | $0 - 10 \frac{J}{mol\,K}$ |

For both networks, the same architecture with the same parameters is used. Using the variables introduced in *Figure 4.14,* the networks' parameters are:

**Table 4.12:** Neural network hyperparameters

| Parameter | Value |
|---|---|
| **N** | 2 |
| **M** | 500 |
| **Output nodes** | 500 |
| **Input nodes** | 1000 |

The training of the network is conducted using the following hyperparameters. As the optimizer, the Adam [33] optimizer is used.

**Table 4.13:** Training hyperparameters

| Parameter | Value |
|---|---|
| **Learning rate** | 0.001 |
| **Number of epochs** | 250 |
| **Batch size** | 1028 |

In the training of either network, the mean squared error is used to evaluate the performance. *Figure 4.20* shows how the training losses develop over the epochs. Both networks show similar behavior, whereas the absolute losses of network 2 are smaller than those of network 1. The losses are calculated in an unsupervised setting: the network outputs its prediction for a function $x^{\alpha} = x^{\alpha}(x^{\beta})$ so that the equations in (*3.29*) are fulfilled. These predictions are plugged into one of the left sides of the equations in (*3.29*). The respective right side is evaluated using the same x-values that are used to create the Gibbs energy function in the training dataset (vector with 500 values between 0 and 1 ordered ascendingly). The loss is then calculated as the deviation of the left side of the equations from the respective right side.

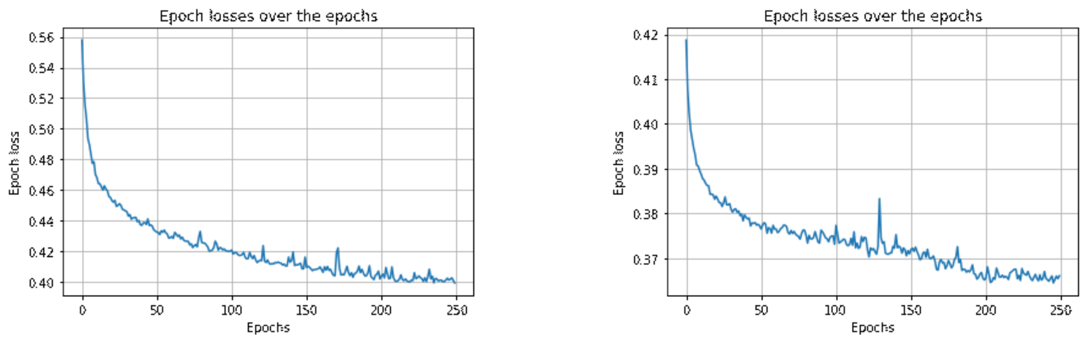


**Figure 4.20:** Training losses of each epoch; left: network 1 (trained to find function $x^{\alpha} = x^{\alpha}(x^{\beta})$ s.t. the slopes are equal), right: network 2 (trained to find function $x^{\alpha} = x^{\alpha}(x^{\beta})$ s.t. the chemical potentials are equal)

## Au-Ag system

Based on the Gibbs energy of the liquid and the solid phase of the Au-Ag system from [32], the phase diagram is calculated. In *Figure 4.21,* the orange and green dots mark the predicted phase diagram, whereas the blue dots are from the actual phase diagram.

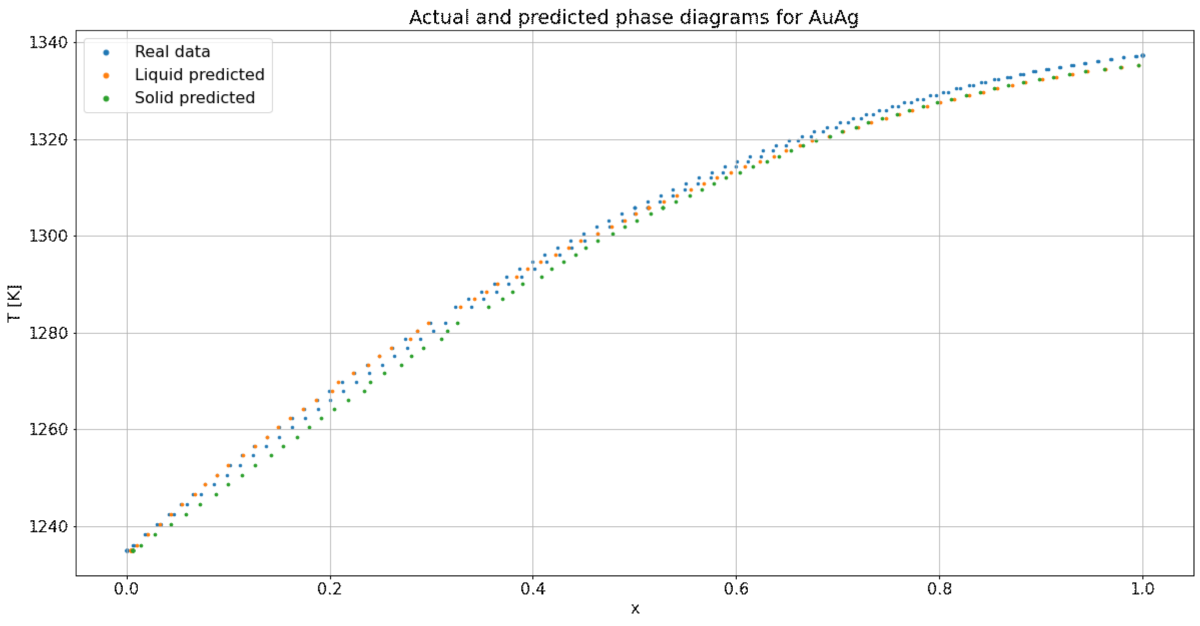


**Figure 4.21:** Predicted and true phase diagrams of the Au-Ag system

The phase diagram can be predicted well. Most notably is the deviation at higher temperatures. To make a comparison between the predicted and the real phase diagram possible, the predictions were made at the temperatures that can be obtained from *ThermoCalc*. *Table* **4.14** presents the results of this comparison.

**Table 4.14:** Statistics of the comparison between predicted and real phase diagram for the Au-Ag system

| Parameter | Value |
|---|---|
| **Mean error** | 0.015655 |
| **Mean squared error** | 0.000493 |
| **Minimum deviation** | 9.7975e-6 |
| **Maximum deviation** | 0.066123 |

The mean execution time of the above-mentioned algorithm for this system is 2.36 seconds with a standard deviation of 71.3ms tested on 7 runs with 20 loops each. Compared to this, the classical approach using numerical techniques (in this case the *SciPy* subpackage *optimize* with the function *least_squares*) takes on average 6.65 seconds with 629ms standard deviation, determined in the same number of loops.

### Lukas-Petzow-Mager functions

The rules for the construction of arbitrary Gibbs energy functions in [12] can be controlled by the parameters $A_{\text{diff}}$ and $A_{\text{sum}}$ (see also chapter *4.4.2*). $A_{\text{diff}}$ can take integer values from $-2$ to 4, whereas $A_{\text{sum}}$ can take the values $-2$, integer values from 0 to 6 as well as 8, 10 and 12. Depending on the choice of these two parameters, Gibbs energy functions can be generated that lead to different kinds of phase diagrams. For further reference, see the original paper [12]. In general, the presented approach aims to predict all phase equilibria. This is although not fully possible, as miscibility gaps in solid phases follow different physical rules than those presented in equation (*3.29*) and can therefore not be calculated with this approach.

In the following figures, a selection of systems will be presented for which the approximation of the phase diagram works well. In the case of $A_{\text{diff}} = 0, A_{\text{sum}} = 0$, the computation using the presented algorithm takes a mean of 3.43 seconds with a standard deviation of 99.3ms, determined over 7 runs with 20 loops each, for a temperature range from 900-1100K (range from one melting point to the other) with a step size of 1K. Compared to this, the classical approach using the *SciPy optimize* package takes on average 24.8 seconds with a standard deviation of 1.4s over the same number of cycles.

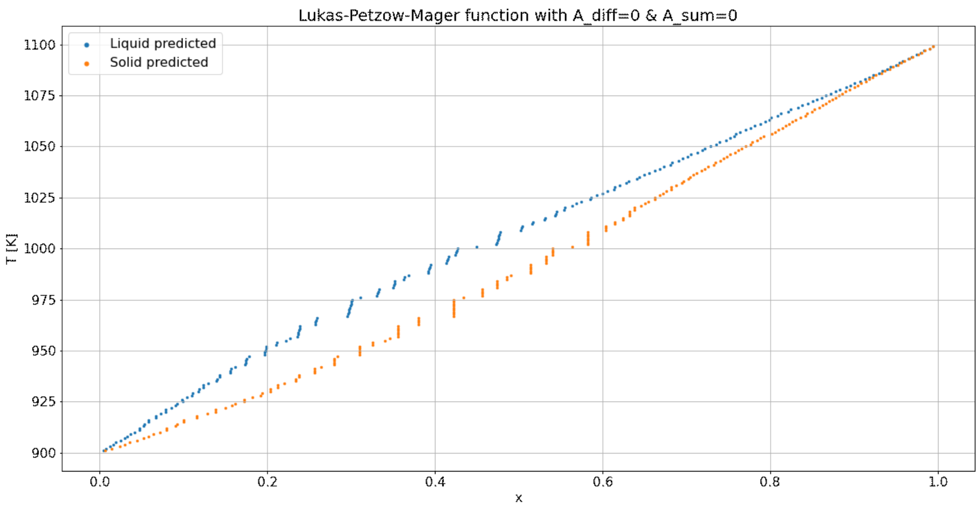


**Figure 4.22:** Predicted phase diagram for $A_{\text{diff}} = 0, A_{\text{sum}} = 0$

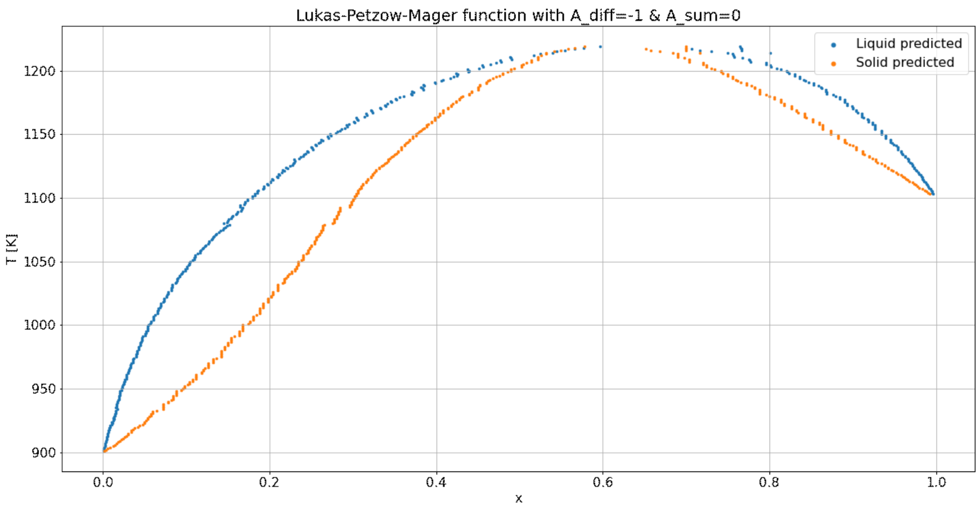


**Figure 4.23:** Predicted phase diagram for $A_{\text{diff}} = -1, A_{\text{sum}} = 0$

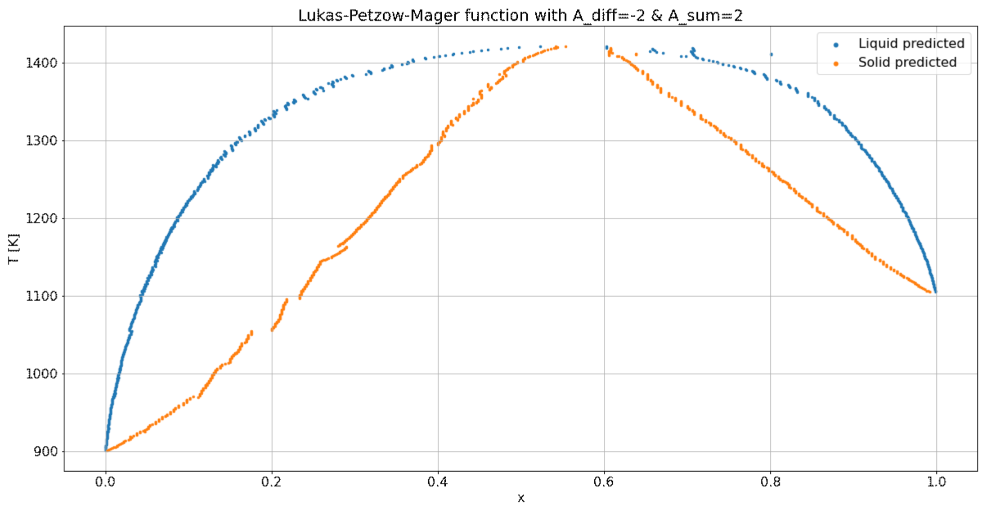


**Figure 4.24:** Predicted phase diagram for $A_{\text{diff}} = -2, A_{\text{sum}} = 2$

Other systems cannot be predicted as well. As *Figure 4.25* shows, for $A_{\text{diff}} = 4, A_{\text{sum}} = 1$, the phase diagram can only be approximated roughly, others, like in *Figure 4.26* for $A_{\text{diff}} = -1, A_{\text{sum}} = 6$, cannot be approximated at all. *Figure 4.27* shows for which systems the phase diagrams can be predicted and for which not. A green symbol means that the phase diagram can be approximated well, yellow means that it can only be approximated roughly, and red means that it cannot be approximated well at all. Invalid combinations of $A_{\text{diff}}$ and $A_{\text{sum}}$ values are marked with no symbol in *Figure 4.27*. For each system, an exemplary phase diagram is shown in the appendix.

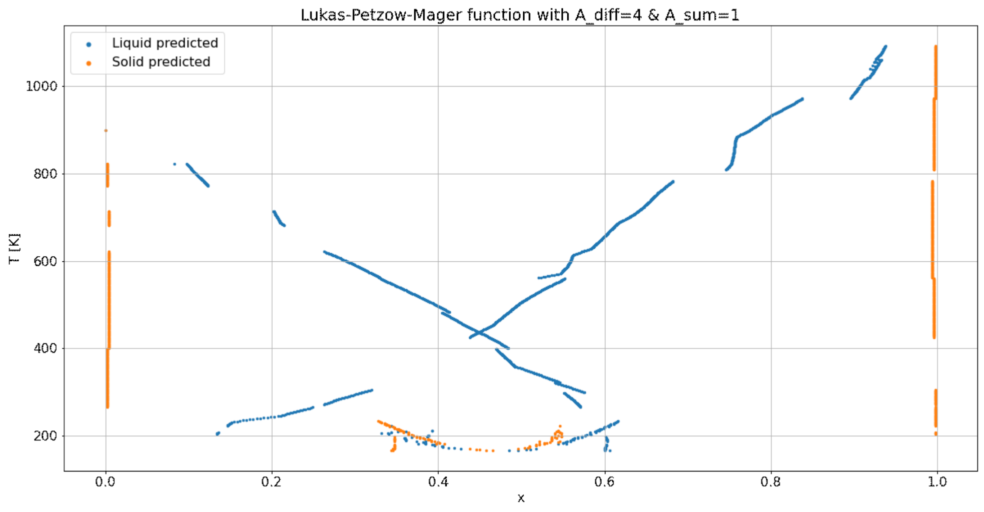


**Figure 4.25:** Predicted phase diagram for $A_{\text{diff}} = 4, A_{\text{sum}} = 1$

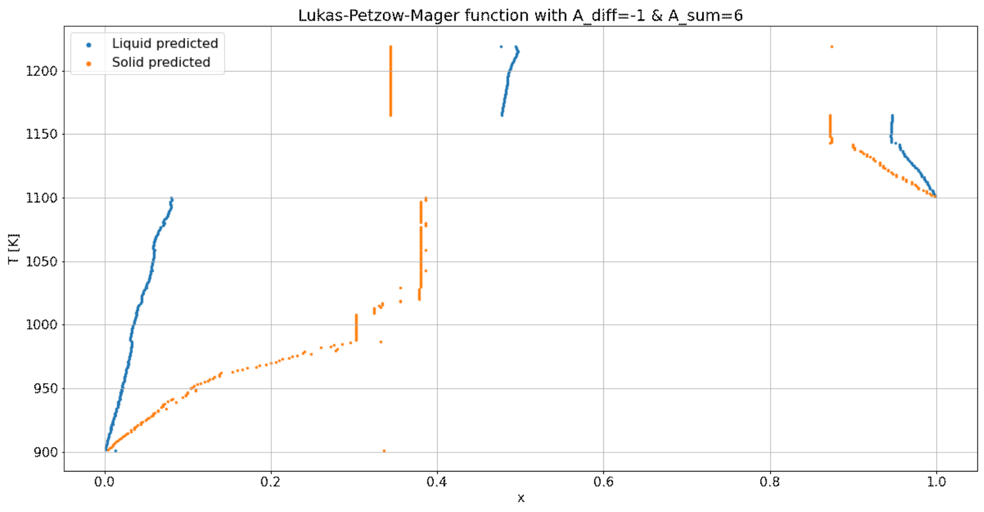


**Figure 4.26:** Predicted phase diagram for $A_{\text{diff}} = -1, A_{\text{sum}} = 6$

| $A_{\text{diff}}$ \ $A_{\text{sum}}$ | -2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 8 | 10 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | ! | ! | ! | ! | ! | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| 3 |  | ! | ! | ! | ! | ! | ✗ | ✗ |  |  |  |
| 2 | ✓ | ! | ! | ! | ! | ! | ✗ | ✗ | ✗ | ✗ | ✗ |
| 1 |  | ✓ | ✓ | ! | ! | ! | ✗ | ✗ |  |  |  |
| 0 | ✓ | ✓ | ✓ | ✓ | ! | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| -1 |  | ✓ | ✓ | ✓ | ! | ! | ✗ | ✗ |  |  |  |
| -2 | ✓ | ✓ | ✓ | ✓ | ✓ | ! | ✗ | ✗ | ✗ | ✗ | ✗ |

**Figure 4.27:** Phase diagrams that can be predicted by the algorithm: green: can be predicted well, yellow: can be predicted roughly, red: cannot be predicted well, no symbol: invalid combination

### 4.4.6 Discussion & Conclusions

#### Discussion

Many of the possible phase diagrams can be predicted well using the presented approach (see *Figure 4.27*). Also, as comparisons with classical optimization approaches show, the neural network works faster. This comes with the drawback that the accuracy is not as good as it is not an accurate calculation method. One of the reasons for this is that the network outputs are only points of a function and not a continuous function, therefore accuracy is lost there. Additionally, the network outputs themselves are only predictions of the desired functions and not analytically computed functions.

Other systems' phase diagrams (see *Figure 4.27*) cannot be approximated at all. This is mostly because they contain miscibility gaps that follow different physical laws than the ones presented in equation (*3.29*). To show that this approach can work, it was assumed to be sufficient to calculate phase diagrams without the miscibility gaps. To also be able to calculate the equilibria in miscibility gaps, the presented approach would have to be changed. A detailed description of this is not given here as this goes beyond the scope of this work.

**Conclusions**

This approach is well-suited to making fast approximations of phase diagrams for many different systems. If accurate phase diagrams are needed, one should use potentially slower numerical methods. To find regions where a phase is stable, this approach is good enough. It is assumed that this approach can work better with more computation capacities at hand. The networks were trained on a regular laptop and using more than 250.000 training samples came close to exceeding the available RAM. With more RAM available, the networks could be trained on a bigger dataset and could therefore generalize better.

It must be noted at this point that the user must provide the correct pairs of Gibbs energy functions to the algorithm. In systems with multiple phases, it might be that there are multiple Gibbs energy functions that, when minimized, lead to different equilibria, although only one (the one with the minimum total Gibbs energy) can be stable. Providing the "wrong" pair of Gibbs energy curves, this algorithm would find the wrong phase equilibria and therefore the wrong phase diagram.

## 4.5 Measurement data classification – thermoclassifier

### 4.5.1 Goal and workflow

The goal of the network implemented in this package is, given a set of pairs of temperature values and measured heat capacity values of any of the 78 elements in the *SGTE* data, to predict both the element which the measurements have been taken from and the phase of every measurement in the set. Not all measurements in the set need to be taken from the same phase, although it is required that they are taken from the same element. The heat capacity is chosen as the property to make the predictions on as it is assumed to be the most likely to be measured in experiments. Although, after sufficient training, the network could theoretically also predict elements and phases based on Gibbs energy, entropy, or enthalpy measurements.

To allow for measurements from different phases in the measurement set, the element and phase classification are split into two separate classification tasks. In case the element and the phases were to be predicted in one single step, it could not be guaranteed that the classifier predicts the same element for every measurement pair in the set. *Figure 4.28* shows the basic workflow for a set of five temperature/heat capacity measurement pairs.

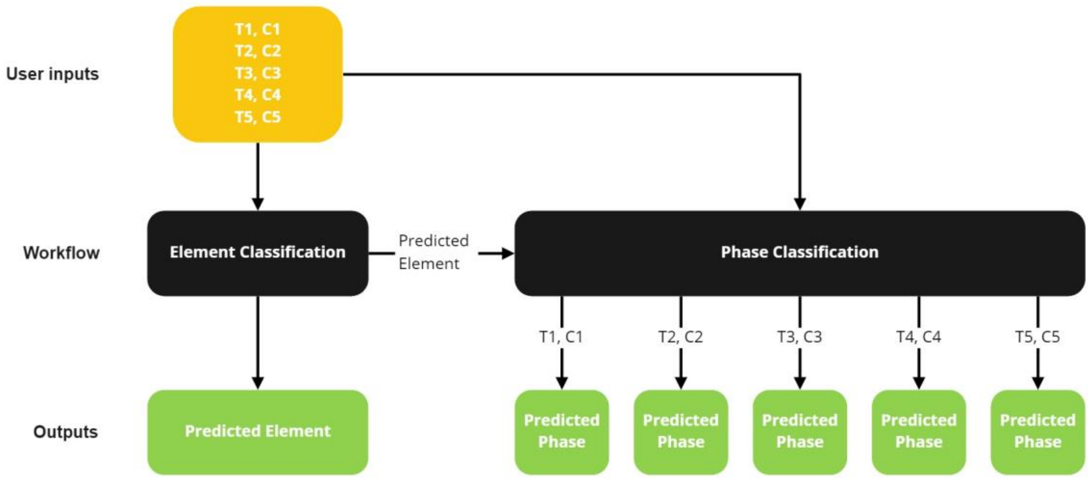


**Figure 4.28:** Basic workflow of the measurement data classification

The network to predict the element takes the set of measurement values as input and gives as output a prediction for the element. The phase classifier then takes the predicted element (in form of a numeric label) and the set of measurement values as the input to predict the phase of every single temperature/heat capacity measurement pair.

### 4.5.2 Dataset logic

For the training and testing of the classification networks, the *SGTE* data is used. The dataset is generated by a *Python* class called *DatasetCreator,* which is part of the *dataset* submodule of the *thermoclassifier* package. The element classifier and the phase classifier are trained individually, therefore also the data for training is generated individually with different settings for the *DatasetCreator. DatasetCreator* takes the following arguments for its definition:

- **Temperature range and step size:** the temperature range which the *SGTE* data is loaded for, and the size of steps taken in this range. For further reference, see *4.1.2.*
- **Measurement type:** the measurement (i.e., either of Gibbs energy, entropy, enthalpy, or heat capacity) to load the data for. As the classification is only made on heat capacity values, the measurement type is always the heat capacity.
- **Sequence length:** number of temperature/heat capacity pairs in each measurement set.
- **Dataset size/splits:** the size of training, test (and, if needed, validation) sets in percent of the size of the whole dataset
- **Validation:** whether to include a validation set
- **Elements:** which elements to load the data for
- **User:** Defines if *DatasetCreator* is used to generate data for the classification of phases or elements.
- **Pressure:** pressure at which the equations are evaluated

For all the elements defined, *DatasetCreator* solves the *SGTE* equations in the defined temperature range, whereas only values from phases stable at the given temperature and pressure are considered as only values from stable phases can be found in real measurement data. It randomly packs temperature/heat capacity value pairs plus a unique label for the element (an integer value in the range from 0 to the number of elements) and a unique label for the phase (just in case of phase classification; integer in the range from 0 to the number of phases of a specific element) in packages of size sequence length. Inside each package, the temperature/heat capacity/label triples are sorted ascendingly by the temperature if the sequence length is greater than 1.

The packages serve as input for the network, whereas the labels are removed and only used to check if the prediction made is correct or not (in the case of phase classification, only the phase label is removed as in this case the element label serves as input to the network). Assuming a temperature range that gives 9 temperature/heat capacity pairs, an arbitrary element with label L, and a sequence length of 3, *Figure 4.29* shows how the data packs are generated for element classification. In the case of phase classification, an additional label for the phase is added.

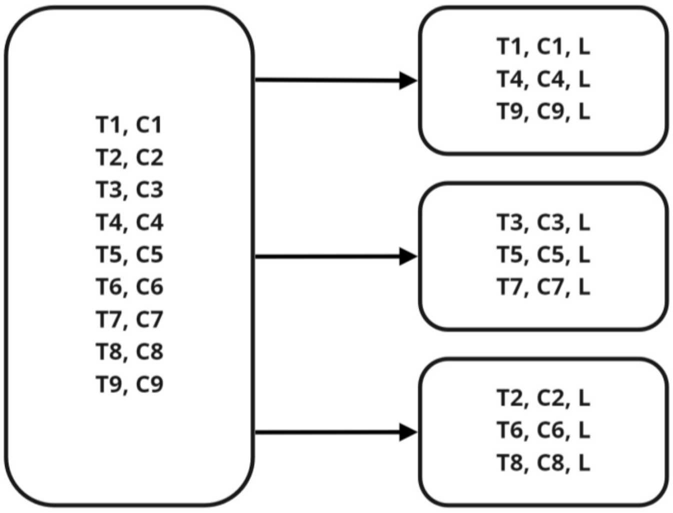


**Figure 4.29:** Packing of data

In *Figure 4.29,* the box on the left contains the values obtained from solving the *SGTE* equations, whereas the temperatures are sorted from T1 to T9 ascendingly. The values are then assigned randomly to a package and passed the label for the element and/or the phase, depending on which classifier is trained. This process is repeated for every element/phase, giving a collection of such packages for every element/phase in the end. For training the networks, the packages are redeemed in random order from this collection so that there is no order anymore.

Additionally, using a class called *TestData*, other datasets can be loaded to check how the classifier reacts to data not from the *SGTE* data. As of *3.2.4,* the data for testing should be generated in the same way as the training data is, which is not the case when using data from other datasets. Although, as this classification network can be used to make predictions on real measurement data, using different sources is a good way to simulate real measurement data. Specifically, the data from *Barin* [15] (see also *3.1.5*) is used for this purpose.

For the training and testing, the data is handed to the network by a *torch DataLoader,* which is part of the *PyTorch* library introduced in *3.2.5*. The *DataLoader* allows to shuffle the data so that in each epoch, the network receives the input in different orderings.

### 4.5.3 Network architectures

As described in *4.5.1,* the task is split into the classification of elements and phases separately, which leads to two individually trained networks. One network is used for the element classification, whereas the other is used for the classification of one element's phases. The element classification network gives as output the most likely element for the given input. This output serves together with the same temperature/heat capacity values the element classifier received as input for the phase classifier.

#### Element classifier

Apart from the number of output classes and therefore the number of output nodes, which is the number of elements in the *SGTE* data (i.e., 78 elements), the network's architecture is designed variable. The number of hidden layers $N$, the number of nodes in the hidden layers $M$ as well as the number of input nodes $2S$ (with $S$ being the sequence length of the *DatasetCreator* introduced in *4.5.2*) can be adapted and are therefore hyperparameters of the network, which means good values must be found for them during training. *Figure 4.30* shows the architecture of the network, which is a fully connected network (i.e., every node of one layer is connected with every node in its neighboring layers), whereas the connections are not shown for clarity (compare with *Figure 3.5.,* which shows all connections). Every hidden layer and the input layer use the ReLU function as their activation function.

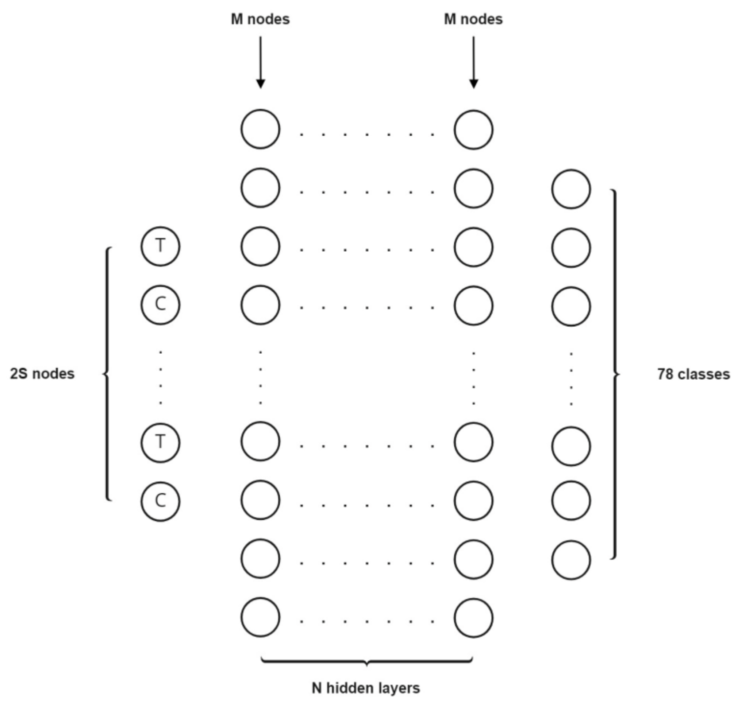


**Figure 4.30:** Network architecture of the element classification

The network receives as input a package of temperature/heat capacity value pairs of size sequence length (in the following referred to as $S$). A linear network layer expects as an input a vector; therefore, the packages are flattened giving a vector of length $2S$. Every element in the input vector corresponds to one input node, whereas due to the package structure the nodes alternatingly receive either a temperature or heat capacity value (see $T$ and $C$ nodes in *Figure 4.30*).

### Phase classifier

The phase classifier's architecture is very similar to the element classifier's, although, in this network, the number of input nodes is not variable as the network makes its predictions on one temperature/heat capacity value pair at a time. Additionally, the network receives an element label as input so that it can correctly assign the measurements. The number of hidden layers $L$ and the number of nodes $K$ in each hidden layer are variable (*Figure 4.31;* again, for clarity shown without the connections between the nodes). This network also uses the ReLU function as the activation function for every node apart from the nodes in the output layer.

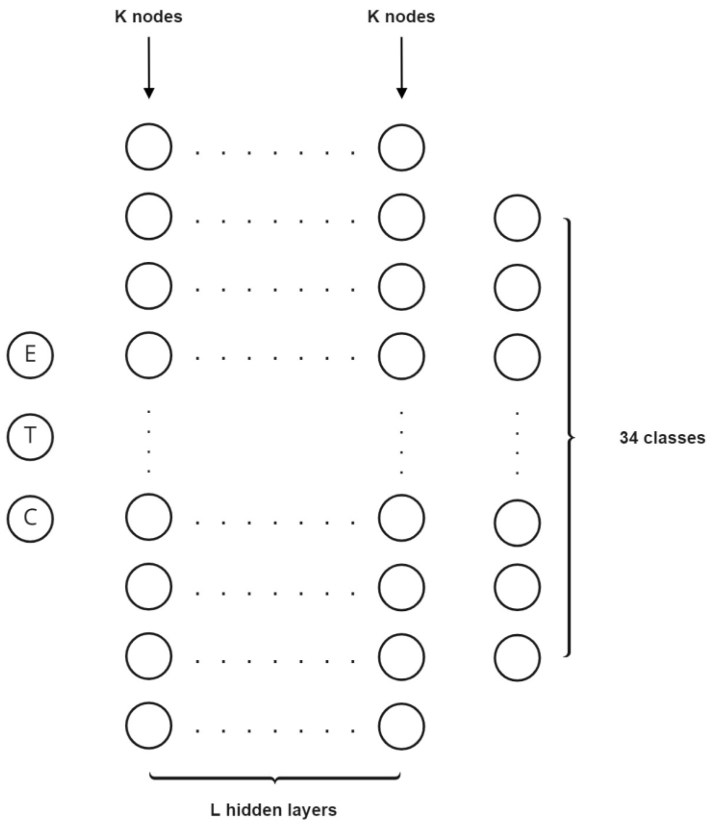


**Figure 4.31:** Network architecture of the phase classification

For the training and testing of the phase classifier, a sequence length of 1 is used for the *Dataset-Creator.* This is because the phase classifier makes one prediction per temperature/heat capacity value pair and not one prediction per pack. Compared to the element classifier, this means that, when the element classifier receives an input with sequence length 5 (5 temperature/heat capacity value pairs), the element classifier makes one prediction on the whole package and the phase classifier makes 5 individual predictions. This is necessary because the measurements are not restricted to being from the same phase.

### 4.5.4 Results

The element and the phase classifier are trained individually, therefore they produce separate results. After sufficient training, the resulting networks are combined into a combined predictor as shown in *Figure 4.28.* In the following, the training and testing results of the individual networks as well as the performance of the combined predictor are presented.

To evaluate the performance of the classification, the prediction accuracy $a$ is used:

$$a = \frac{\text{correct predictions}}{\text{correct predictions} + \text{false predictions}}$$

For the element classifier, two different approaches regarding the training data are made. First, the classifier is trained on the plain *SGTE* data as obtained from evaluating the equations. This gives data points that lie on a smooth curve without any variance. As real measurement data, in general, has variance in it, the classifier is also trained with random normal distributed noise added to the *SGTE* data.

The phase classifier is trained only on the plain *SGTE* data as in this case variance in the data makes no difference because the phase can be distinguished by the element and the temperature. The heat capacity value is therefore additional information.

Instead of the simple gradient descent method described in *3.2.3,* the *Adam* optimizer introduced in [33] is used. The *Adam* optimizer uses adaptive learning rates and claims to have little memory requirements [33]. The detailed algorithm is irrelevant to this work but can be found in the publication. As the loss function, the cross-entropy loss (equation *3.32*) is used for both the element and the phase classifiers.

#### Element classifier – plain SGTE data

**Accuracy on *SGTE* data test set:** 97.82%

**Accuracy on *Barin* test set:** 62.04%

For the training of this classifier, the (hyper-) parameters shown in *Table 4.15, Table 4.16,* and *Table 4.17* are used:

**Table 4.15:** *DatasetCreator* hyperparameters

| Parameter | Value |
|---|---|
| **Temperature range** | 200K-2000K |
| **Temperature step size** | 0.05K |
| **Sequence length S** | 5 |
| **Training-/test set splits** | 80%/20% |
| **Pressure** | 1 bar |

With the hyperparameters chosen for the *DatasetCreator* (*Table 4.15*), the training dataset has a shape of (411319, 5, 3). There, 411319 is the number of packs with 5 temperature/heat capacity/label triples (see also *4.5.2*).

**Table 4.16:** Neural network hyperparameters (see also *Figure 4.30*)

| Parameter | Value |
|---|---|
| **2S** | 10 |
| **M** | 128 |
| **N** | 2 |

A more complex network with five hidden layers ($N = 5$) does not increase the accuracy when tested on the *SGTE* data test set as it achieves the same result. Although, with an accuracy of 64.6%, it achieves slightly better results when tested on the *Barin* data.

**Table 4.17:** Hyperparameters for the training routine

| Parameter | Value |
|---|---|
| **Learning rate** | 0.001 |
| **Number of epochs** | 250 |
| **Batch size** | 256 |

The chosen learning rate of 0.001 proves to be optimal for this task. Higher learning rates result in a not converging training routine. This means, that the average loss over one epoch does not constantly decrease over time but rather decreases and increases alternatingly, i.e., the losses start to oscillate.

Training for more epochs does not lead to better results. The same network trained for 500 instead of 250 epochs achieves an accuracy of 98.06% on the *SGTE* data test set and 61.31% on the *Barin* data. *Figure 3.1* confirms this fact, as after around 300 epochs no real progress in neither the minimization of the loss nor the increase of the training accuracy (on the *SGTE* data training set) can be seen. Although, already after a few epochs, the training accuracy is above 90%.

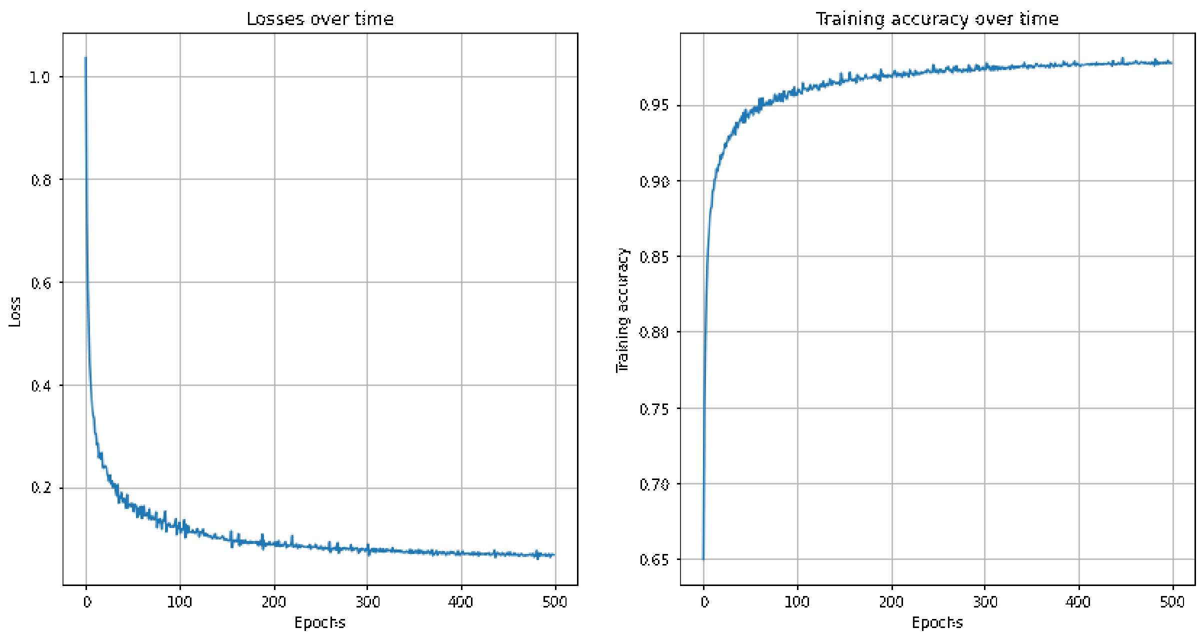


**Figure 4.32:** *Left:* Mean loss per epoch over all epochs, *right:* Training accuracies over all epochs

Also, an increase in the sequence length does not increase the accuracy. Trained on a sequence length of 10, the classifier achieves an accuracy of 95.01% on the *SGTE* data test set. For practical use, a shorter sequence length is also advantageous, as a longer sequence length means that more measurements are required. To make predictions, it is possible to give fewer measurements than the sequence length trained on as input, although this also leads to worse prediction accuracy as information important for the network is not included.

### Element classifier – SGTE data with variance

**Accuracy on *SGTE* data test set:** 91.02%

**Accuracy on *Barin* test set:** 59.12%

Evaluating the *SGTE* data gives data points where all values are on smooth curves with no variance around these curves. For real measurement data, it must be assumed that this is not the case. Therefore, for the training of this classifier random normal distributed noise is added on top of the *SGTE* data. Before every epoch, a new noise vector is generated and added to the data so that the variance effect is maximized.

The classifier is trained using the same hyperparameters as shown in *Table 4.15, Table 4.16,* and *Table 4.17.*

For the training of the results presented above, the normal distribution $\mathcal{N}(0,0.75)$ is used to add the random noise. Increasing the standard deviation leads to lower accuracy on both the *SGTE* data test set and the *Barin* data. With the normal distribution $\mathcal{N}(0,1.25)$, an accuracy of 84.81% on the *SGTE* data test set and 54.38% on the *Barin* data can be achieved. Training the network for more epochs does not lead to better results, as a network trained with $\mathcal{N}(0,0.75)$ for 500 epochs achieves 89.12% accuracy on the *SGTE* data test set and 58.39% on the *Barin* data.

### Phase classifier

**Accuracy on *SGTE* data test set:** 95.63%

As the information about the phases in the *Barin* tables is not as extensive as it is in the *SGTE* data, testing the phase classifier on the *Barin* data is not possible because of lacking phase labels. For the training of this classifier, the (hyper-) parameters shown in *Table 4.18, Table 4.19,* and *Table 4.20* are used:

**Table 4.18:** *DatasetCreator* hyperparameters

| Parameter | Value |
|---|---|
| **Temperature range** | 200K-2000K |
| **Temperature step size** | 0.5K |
| **Sequence length S** | 1 |
| **Training-/test set splits** | 80%/20% |
| **Pressure** | 1 bar |

With the hyperparameters chosen for the *DatasetCreator* (*Table 4.15*), the training dataset has a shape of (205592, 1, 4). There, 205592 is the number of packs with 1 temperature/heat capacity/element label/phase label quadruples (see also *4.5.2*). The step size is just 0.5K as finer steps lead to bigger datasets and problems with the working memory while training.

**Table 4.19:** Neural network hyperparameters (see also *Figure 4.30*)

| Parameter | Value |
|---|---|
| **K** | 128 |
| **L** | 2 |

**Table 4.20:** Hyperparameters for the training routine

| Parameter | Value |
|---|---|
| **Learning rate** | 0.001 |
| **Number of epochs** | 250 |
| **Batch size** | 256 |

Also, a less complex network with $K = 32$ and $L = 1$ was tested. This led to an accuracy of just 87.78% and needed 5000 training epochs to reach this performance.

### Combined classifier

**Element prediction accuracy:** 97.99%

**Phase prediction accuracy:** 94.46%

**Combined prediction accuracy:** 93.49%

The combined classifier makes predictions on both the element and the phase using the workflow shown in *Figure 4.28*. It has no own network but rather uses the fully trained element and phase classifiers presented above. The element classifier used is the one trained on the plain *SGTE* data. For the evaluation of the combined classifier, three metrics are used: the accuracy of element predictions, the accuracy of phase predictions, and the combined accuracies. Whilst the accuracy of element and phase predictions are the same as the accuracy used when training the two classifiers, the values are almost the same as after training. The differences occur because, for the testing of the individual networks and the testing of the combined classifier, different data packages are used as they are always randomly generated. For the combined accuracy, only the predictions where both the element and the phase are classified correctly are viewed as correct predictions. The accuracies are only evaluated on the *SGTE* data set as the phase classification is not possible with the *Barin* data. For the *DatasetCreator* and the data loading, the following hyperparameters are used for the testing dataset:

**Table 4.21:** *DatasetCreator* hyperparameters

| Parameter | Value |
|---|---|
| **Temperature range** | 200K-2000K |
| **Temperature step size** | 1K |
| **Sequence length S** | 5 |
| **Training-/test set splits** | 100%/0% |
| **Pressure** | 1 bar |
| **Batch size** | 256 |

The sequence length is 5 as the input is first passed to the element classifier, which is trained on a sequence length of 5. After the predictions of the element classifier, the data packs are split so that the phase classifier receives the expected sequences of length 1. Also, for this case, no training and test set split is necessary as the network is only evaluated and not furtherly trained. Therefore, all the data generated go into the training set which is used to evaluate the network.

### 4.5.5 Discussion & Conclusions

#### Discussion

All the networks presented above can classify heat capacity data from the *SGTE* database well. Depending on the network design and data preprocessing, some networks perform better than others, in general, although, all of them score a test accuracy that can be considered good. It must be noted that this network was trained on the *SGTE* data and will therefore perform better on data that is similar to the *SGTE* data. In general, variance is expected when conducting real-world measurements. Therefore, the measurement data will slightly vary from the *SGTE* model. To account for this, the network was tested on the data published in [15]. This data differs from the *SGTE* data, but still, an accuracy of around 60% could be achieved. The lower accuracy comes from the fact that the curves in the *SGTE* data and the ones presented in [15] are not the same. As the network is trained on 78 elements, this fact leads to misclassifications.

Integrating different databases in the training dataset is an option. The problem although is that to be able to classify all 78 elements in the *SGTE* data, a vast amount of training data is necessary. Publications such as [15] although provide only small numbers of data points for each element, whereas the *SGTE* database provides equations of the temperature which can be evaluated at infinitely many points. Therefore, the creation of infinitely large datasets is theoretically possible. Compared to the big number of data points drawn from the *SGTE* database, far smaller numbers of data points from other sources would not have a big effect on the outputs of the networks.

#### Conclusion

This network is suitable for real-world applications to classify measurement data quickly. Depending on the measurement accuracy as well as how similar the measurement data points are to the *SGTE* model, the predictions' accuracy will vary. To improve the generalization abilities of the network, it is possible to retrain a new network using the same design but include data from different sources or use the existing network and try to improve it by retraining it using a more general dataset. As described above, this will only lead to satisfactory results if the available amount of data from other sources is as vast as the amount of data from the *SGTE* database.

# 5 General discussion, conclusions & outlook

## General discussion and conclusions

Apart from the rebuilding of the *LaengeNet* presented in *4.2,* all presented machine learning applications delivered good results both during training and testing.

Depending on the problem, a choice must be made whether machine learning or classical approaches make more sense. In the case of binary phase equilibria prediction, the presented approach works well when predictions are needed quickly. It is in general although not sufficient when accurate results are needed. The *ThermoNet* and the *LaengeNet* can help to obtain thermodynamic functions for unary systems. Using the obtained thermodynamic functions, phase equilibria in unary systems can be calculated. This is only useful if no analytic functions for the thermodynamic quantities are available. For example, all the equilibria of pure elements contained in the *SGTE* database are easier and faster obtained directly from the database than making the detour using a neural network approximation. Lastly, the element and phase classifier can be helpful if measurements are made on a sample where the actual element and the phase have to be determined (which however must be part of the *SGTE* data). Depending on the accuracy of the measurements, a good prediction on both the element and the phase can be made. In case the element is known but not the phase, this classifier can also be used to only classify the phases.

## Outlook

In the future, neural networks will be used in thermodynamics in the above presented as well as in other areas. The binary equilibrium prediction is promising but will have to be improved using more computation capacity if it was to be used in real-world applications. The approach chosen is very general so that it can, in theory, approximate phase diagrams of all different sorts. In a specific use case (for example in steel production), where the chemical system is always roughly the same, it is better to train a network specifically on this system. For this case, very good results are expected. The approach for binary equilibrium calculation presented in this work might be enhanced by giving the coefficients of approximating polynomials rather than discrete values of the functions as outputs of the network (compare with *Figure 4.17*). The problem with this idea is that in general, the output of either of the networks can contain the values of multiple functions (in case one or both Gibbs energy functions have inflection points). Therefore, the values cannot be approximated by one polynomial. An additional unit, which determines the number of functions, would be needed. It is assumed that this approach would lead to more accurate results, the problem however is that the task of determining how many functions occur is not trivial.

Also, the *ThermoNet/LaengeNet* can have use-cases in the future. Modeling material given measurement data is the most realistic application. Both those networks are only trained for iron, but given physical laws, it can be possible to find a general network that can predict a material model for any element/unary system.

The element and phase classification can also have real-world use cases. Apart from the one presented, where the element and the phase are predicted given heat capacity/temperature measurement pairs, also phase predictions for binary or multi-component systems based on temperature and compositions are possible applications. Especially the latter, if trained on a specific system can be of help if quick decisions on which phase is present is needed.

# 6 Appendix

## 6.1 Predicted phase diagrams for Lukas-Petzow-Mager functions

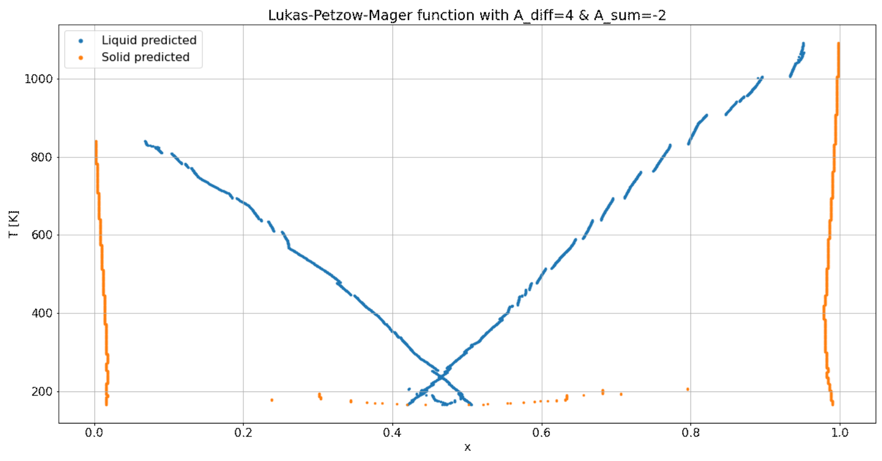


**Figure 6.1:** Predicted phase diagram for $A_{\text{diff}} = 4, A_{\text{sum}} = -2$

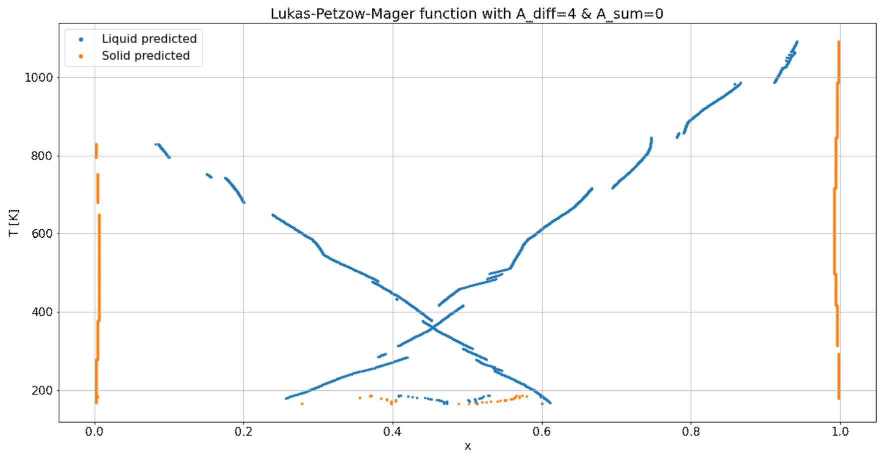


**Figure 6.2:** Predicted phase diagram for $A_{\text{diff}} = 4, A_{\text{sum}} = 0$

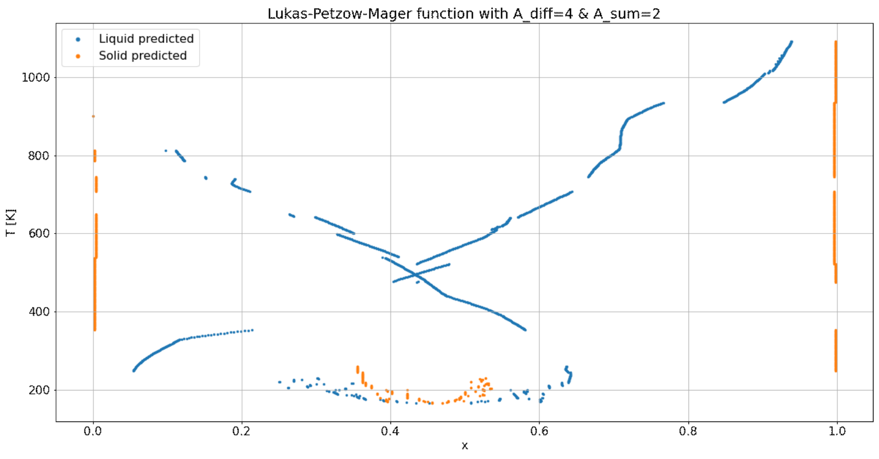


**Figure 6.3:** Predicted phase diagram for $A_{\text{diff}} = 4, A_{\text{sum}} = 2$

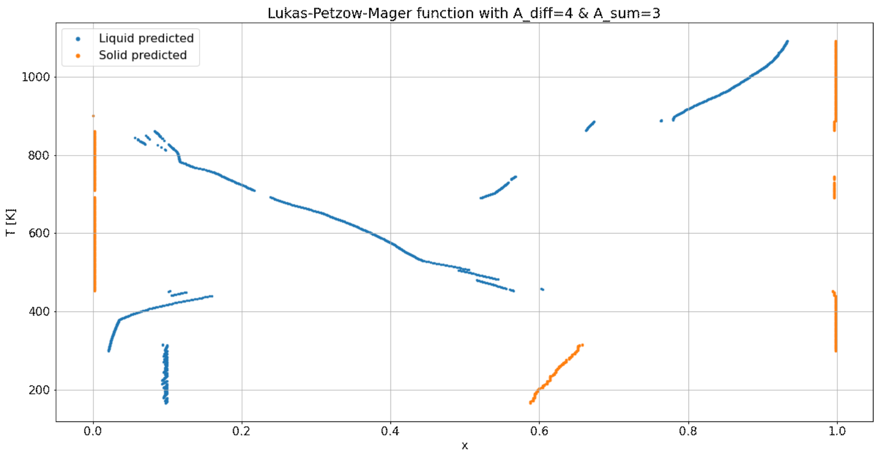


**Figure 6.4:** Predicted phase diagram for $A_{\text{diff}} = 4, A_{\text{sum}} = 3$

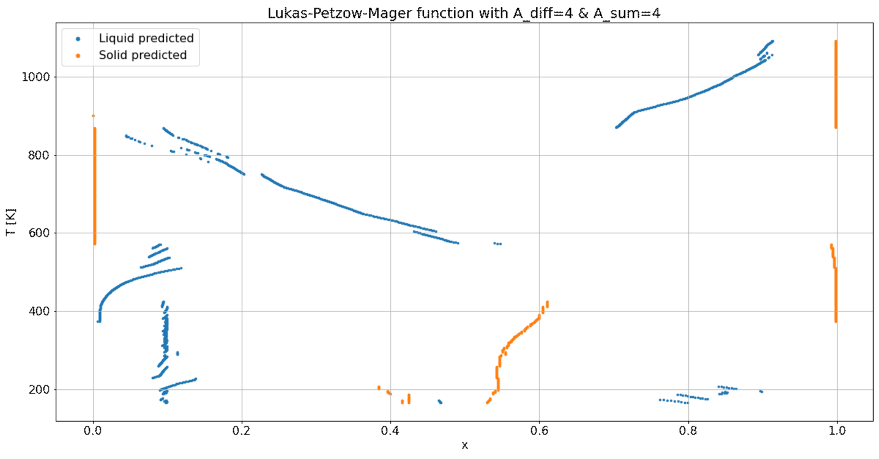


**Figure 6.5:** Predicted phase diagram for $A_{\text{diff}} = 4, A_{\text{sum}} = 4$

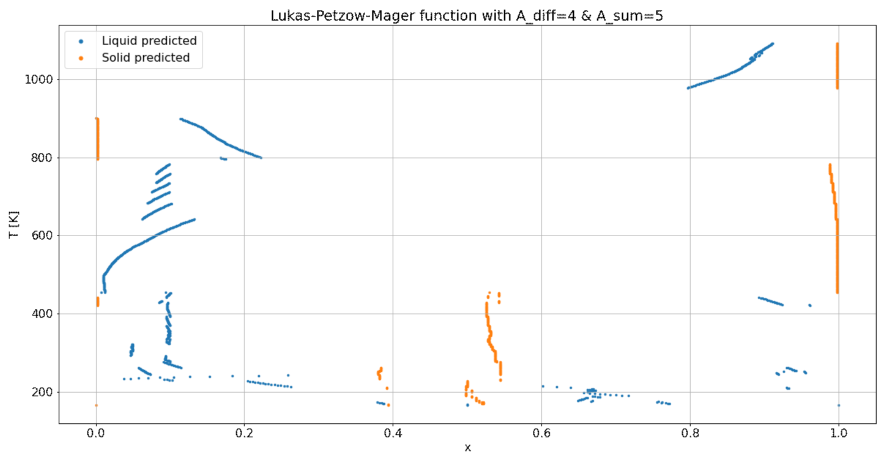


**Figure 6.6:** Predicted phase diagram for $A_{\text{diff}} = 4, A_{\text{sum}} = 5$

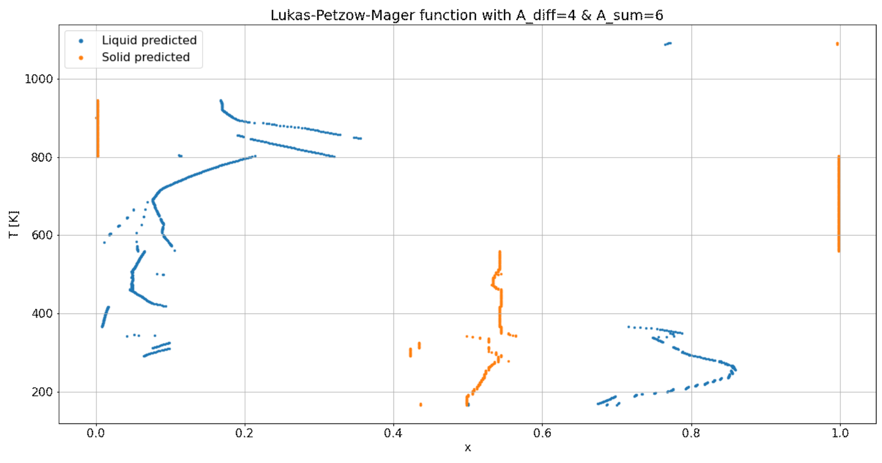


**Figure 6.7:** Predicted phase diagram for $A_{\text{diff}} = 4, A_{\text{sum}} = 6$

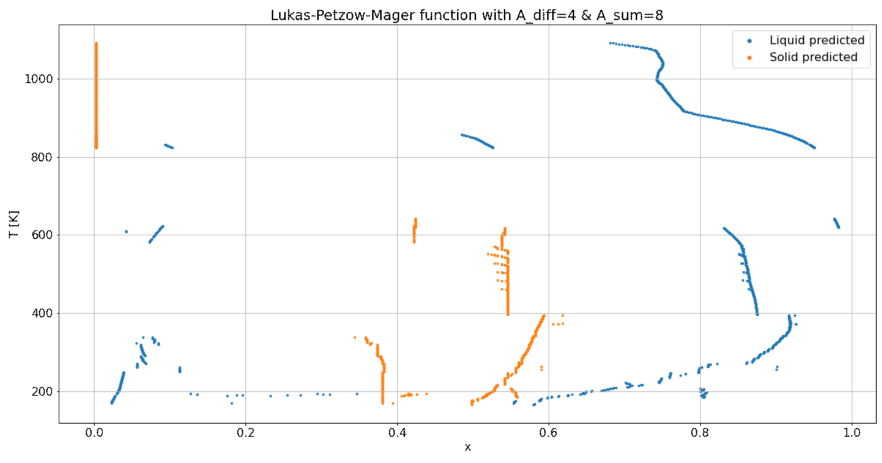


**Figure 6.8:** Predicted phase diagram for $A_{\text{diff}} = 4, A_{\text{sum}} = 8$

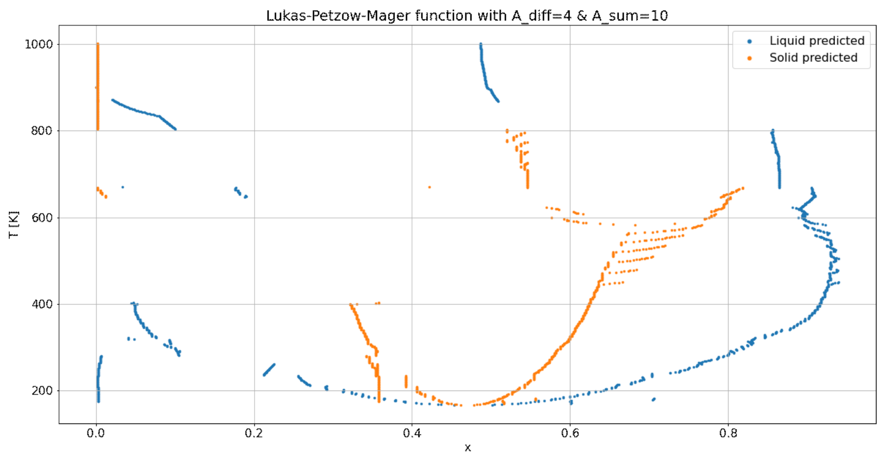


**Figure 6.9:** Predicted phase diagram for $A_{\text{diff}} = 4, A_{\text{sum}} = 10$

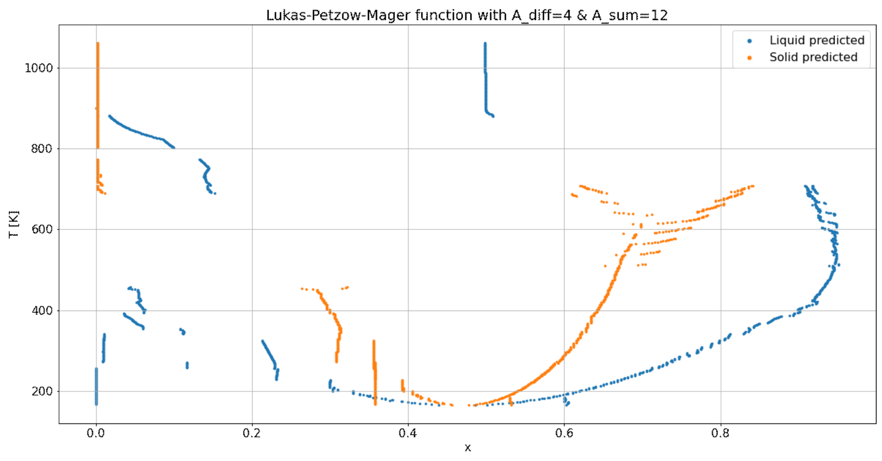


**Figure 6.10:** Predicted phase diagram for $A_{\text{diff}} = 4, A_{\text{sum}} = 12$

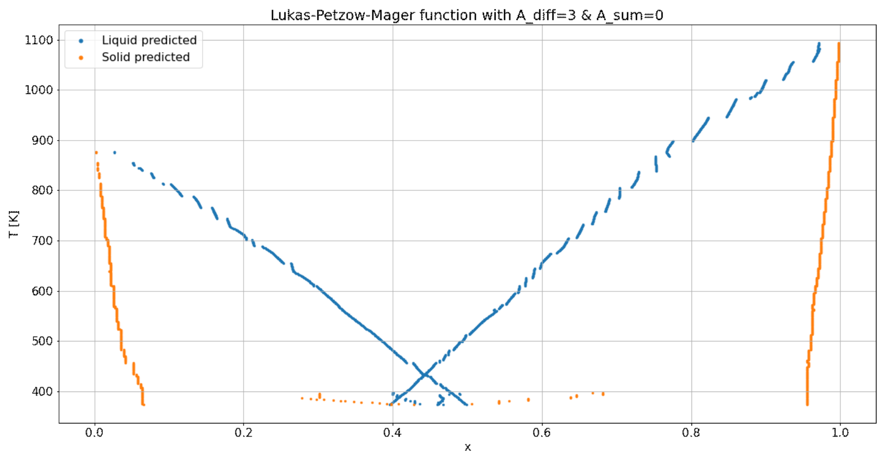


**Figure 6.11:** Predicted phase diagram for $A_{\text{diff}} = 3, A_{\text{sum}} = 0$

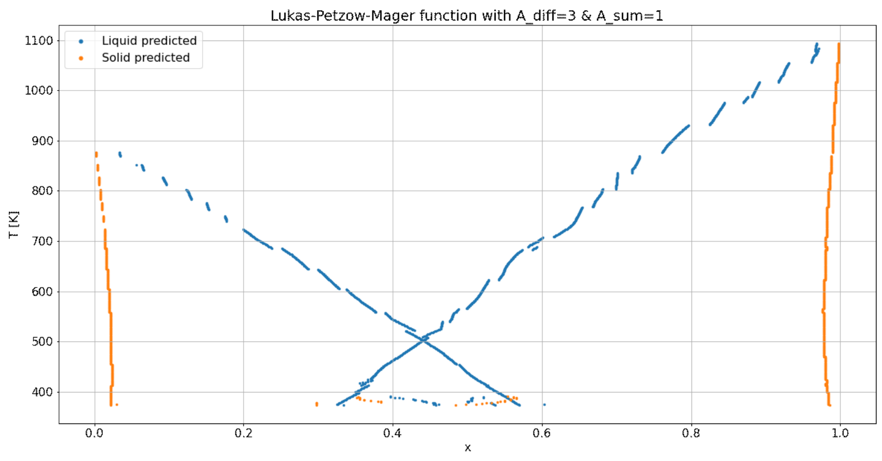


**Figure 6.12:** Predicted phase diagram for $A_{\text{diff}} = 3, A_{\text{sum}} = 1$

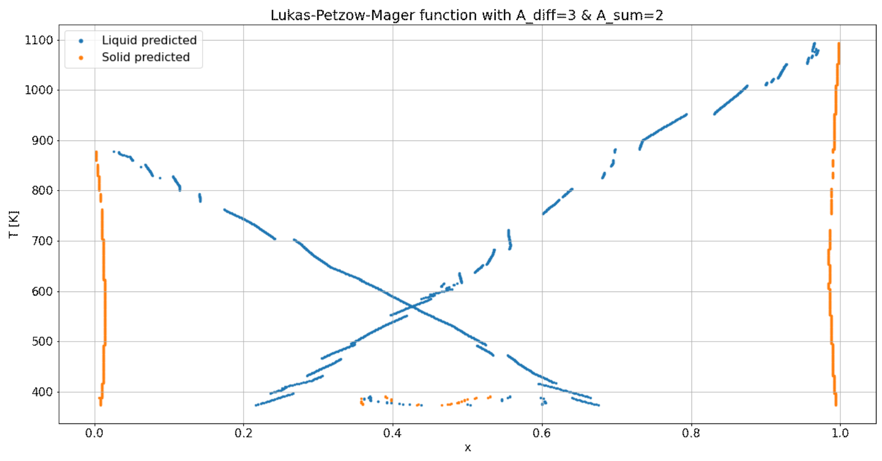


**Figure 6.13:** Predicted phase diagram for $A_{\text{diff}} = 3, A_{\text{sum}} = 2$

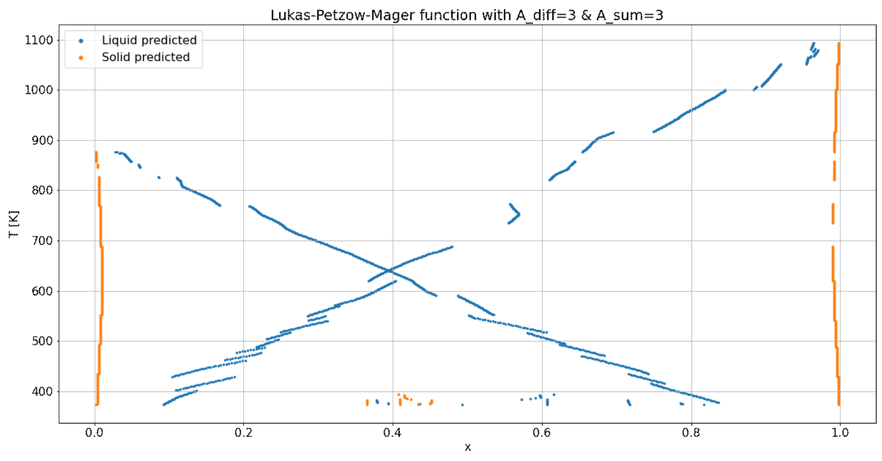


**Figure 6.14:** Predicted phase diagram for $A_{\text{diff}} = 3, A_{\text{sum}} = 3$

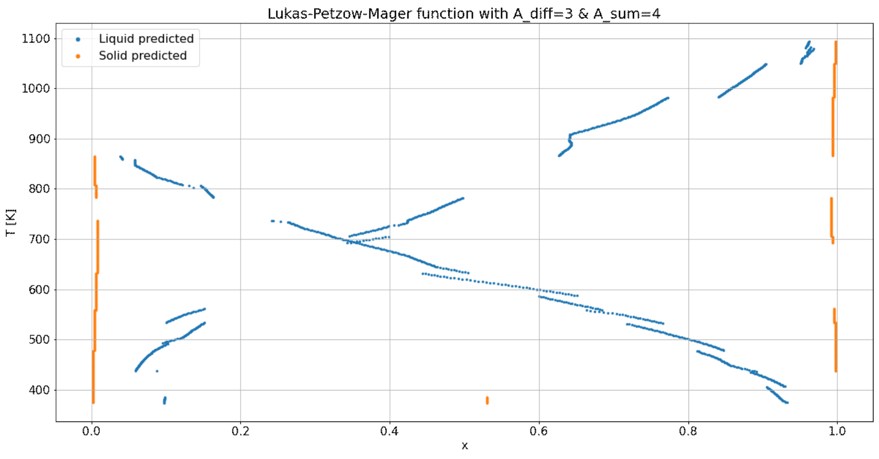


**Figure 6.15:** Predicted phase diagram for $A_{\text{diff}} = 3, A_{\text{sum}} = 4$

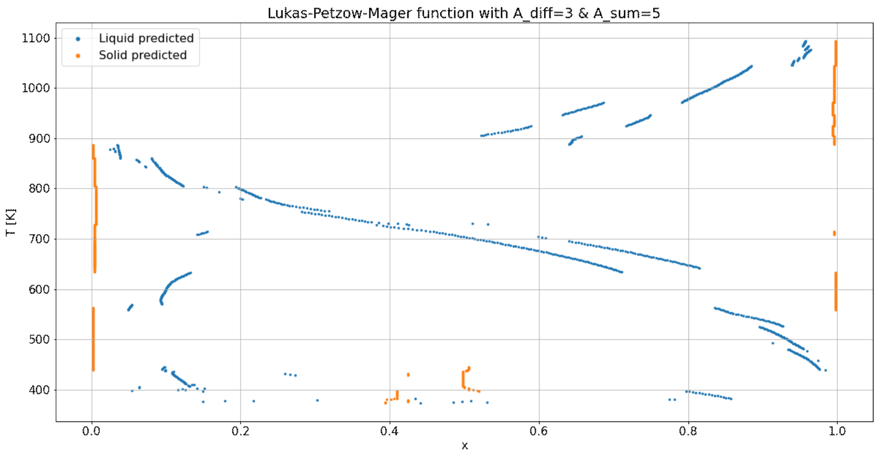


**Figure 6.16:** Predicted phase diagram for $A_{\text{diff}} = 3, A_{\text{sum}} = 5$

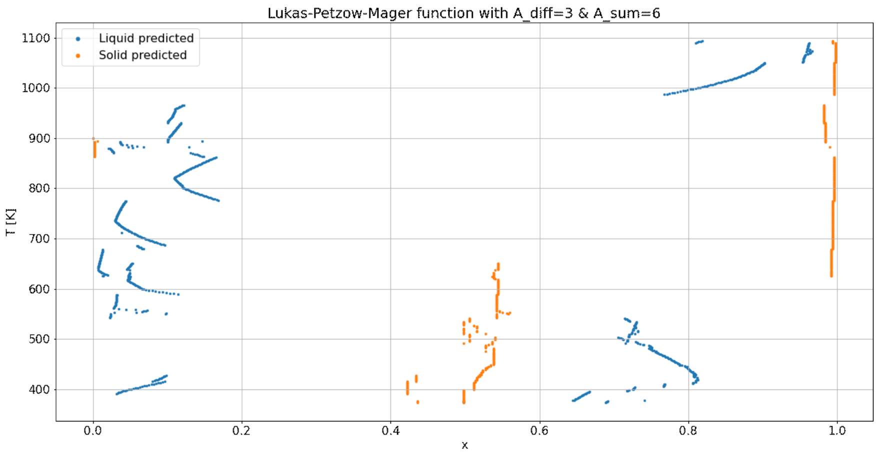


**Figure 6.17:** Predicted phase diagram for $A_{\text{diff}} = 3, A_{\text{sum}} = 6$

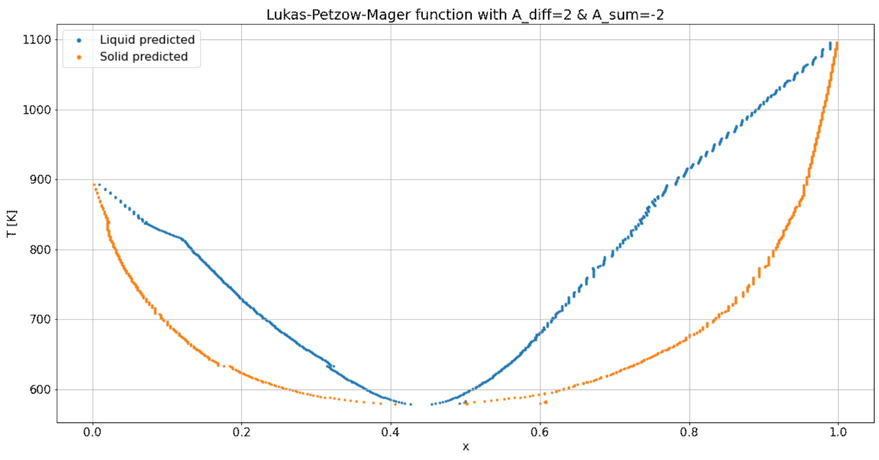


**Figure 6.18:** Predicted phase diagram for $A_{\text{diff}} = 2, A_{\text{sum}} = -2$

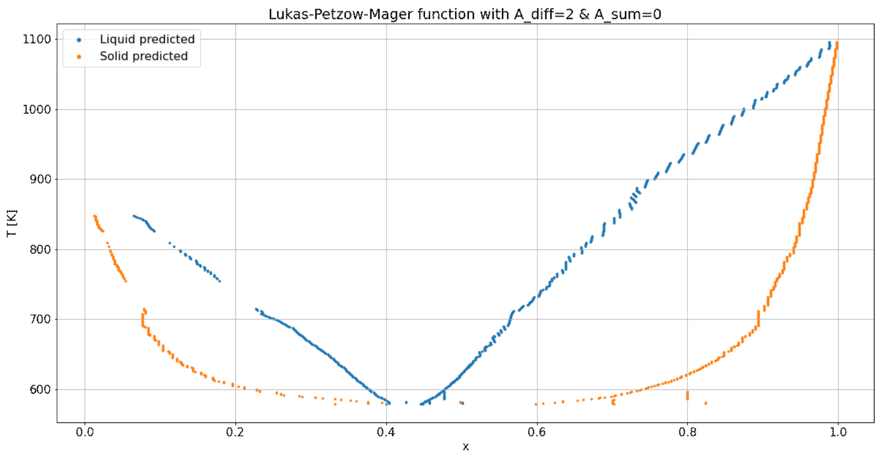


**Figure 6.19:** Predicted phase diagram for $A_{\text{diff}} = 2, A_{\text{sum}} = 0$

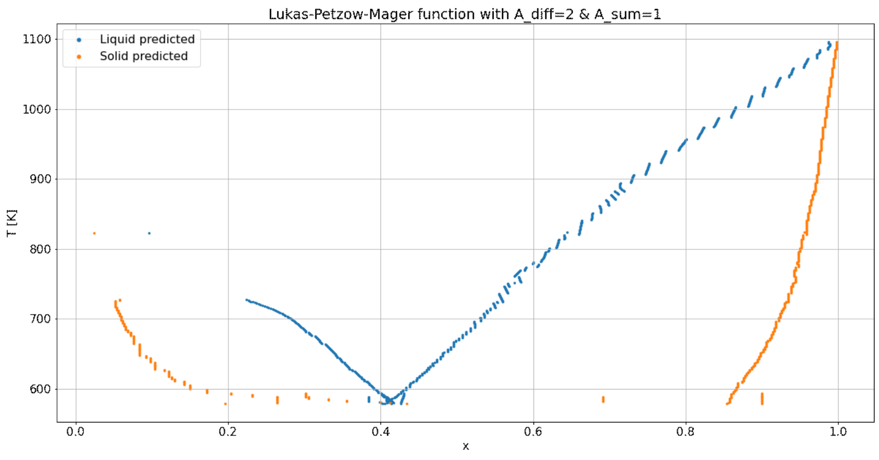


**Figure 6.20:** Predicted phase diagram for $A_{\text{diff}} = 2, A_{\text{sum}} = 1$

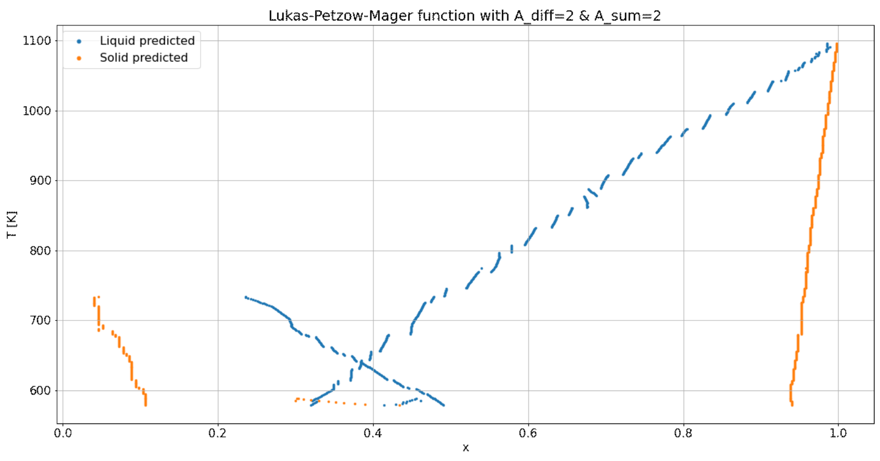


**Figure 6.21:** Predicted phase diagram for $A_{\text{diff}} = 2, A_{\text{sum}} = 2$

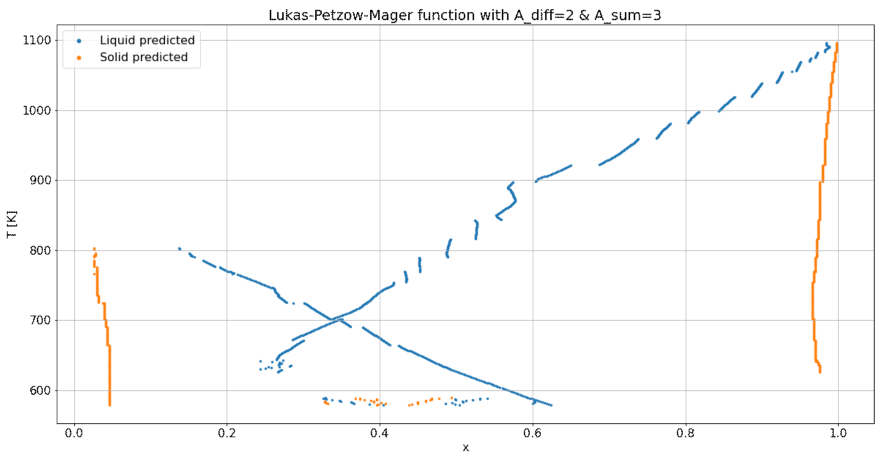


**Figure 6.22:** Predicted phase diagram for $A_{\text{diff}} = 2, A_{\text{sum}} = 3$

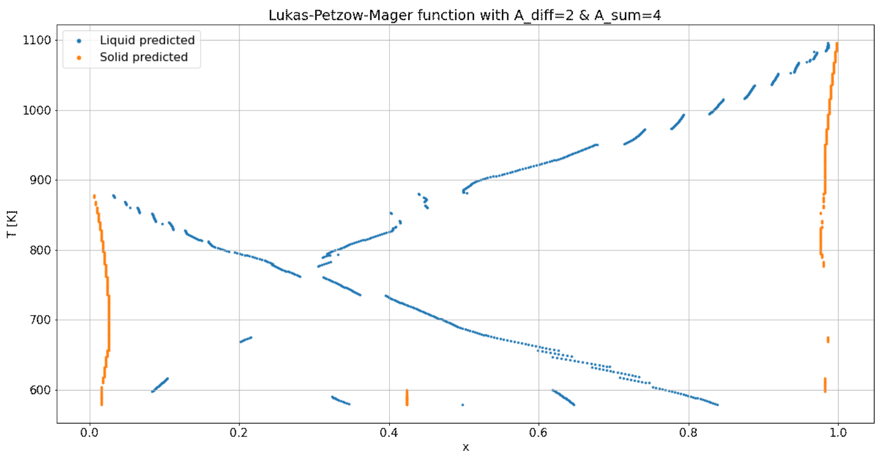


**Figure 6.23:** Predicted phase diagram for $A_{\text{diff}} = 2, A_{\text{sum}} = 4$

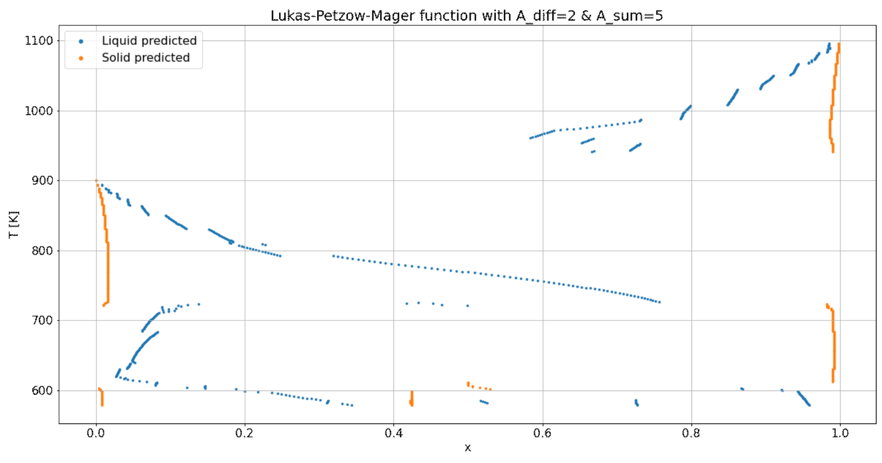


**Figure 6.24:** Predicted phase diagram for $A_{\text{diff}} = 2, A_{\text{sum}} = 5$

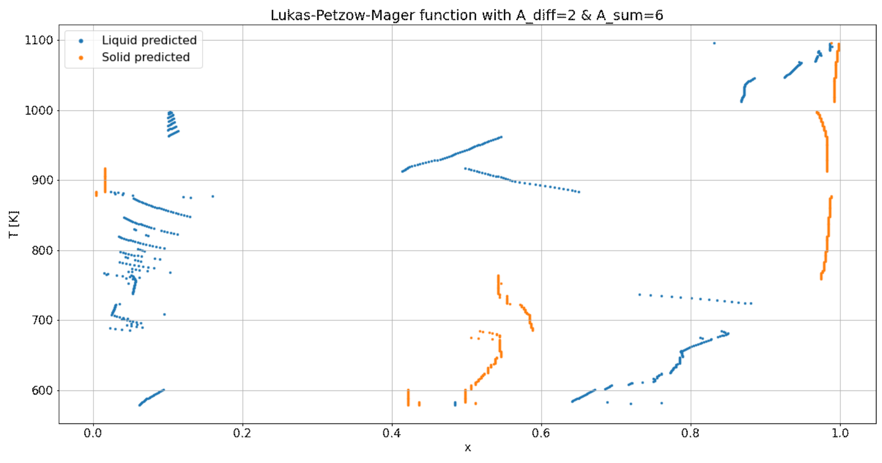


**Figure 6.25:** Predicted phase diagram for $A_{\text{diff}} = 2, A_{\text{sum}} = 6$

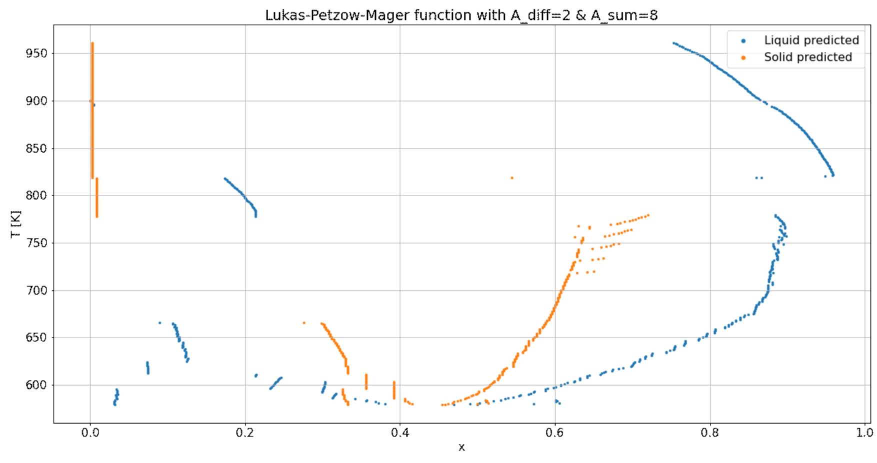


**Figure 6.26:** Predicted phase diagram for $A_{\text{diff}} = 2, A_{\text{sum}} = 8$

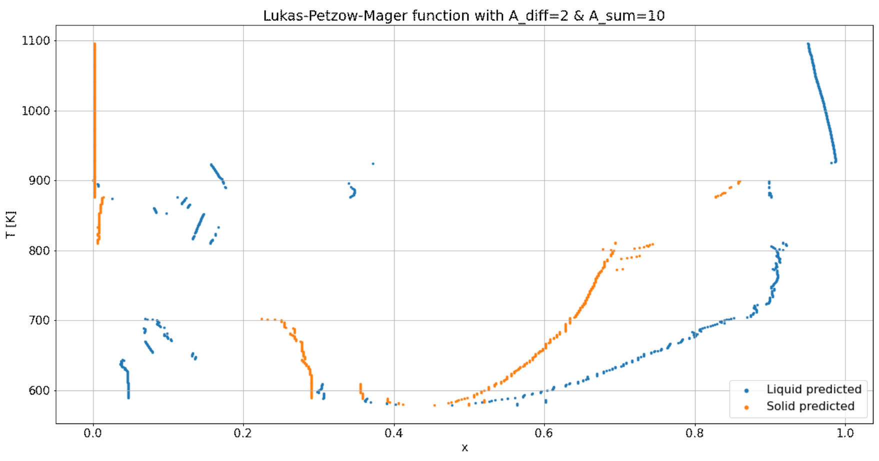


**Figure 6.27:** Predicted phase diagram for $A_{\text{diff}} = 2, A_{\text{sum}} = 10$

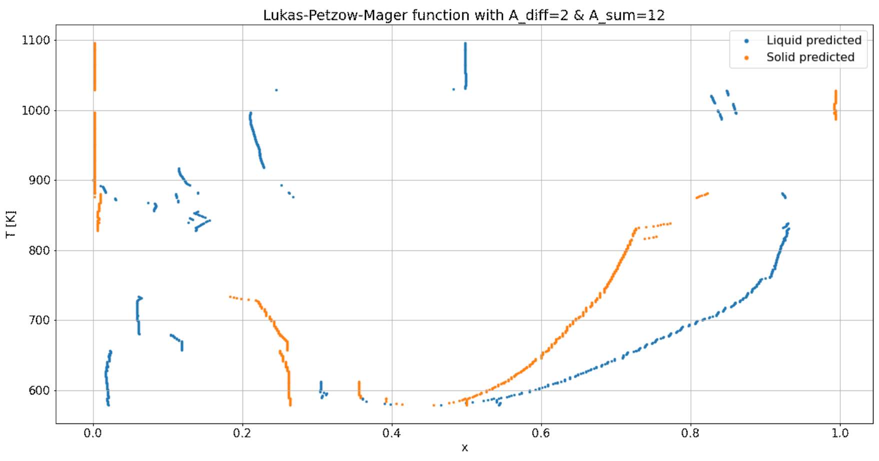


**Figure 6.28:** Predicted phase diagram for $A_{\text{diff}} = 2, A_{\text{sum}} = 12$

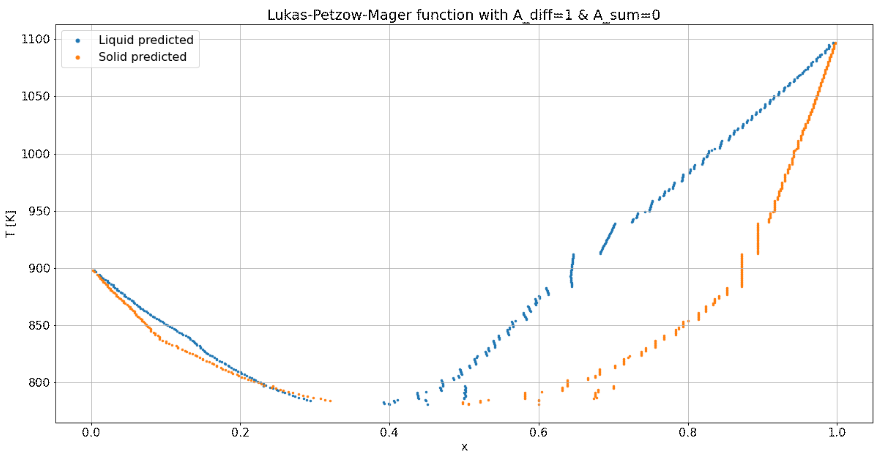


**Figure 6.29:** Predicted phase diagram for $A_{\text{diff}} = 1, A_{\text{sum}} = 0$

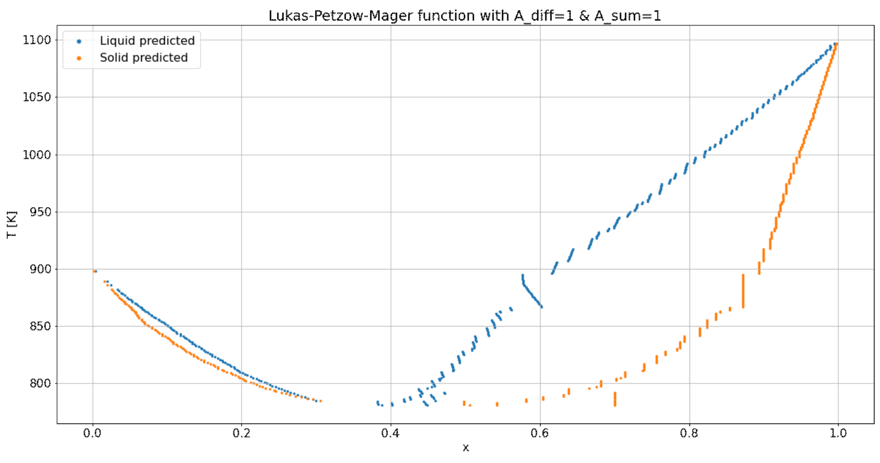


**Figure 6.30:** Predicted phase diagram for $A_{\text{diff}} = 1, A_{\text{sum}} = 1$

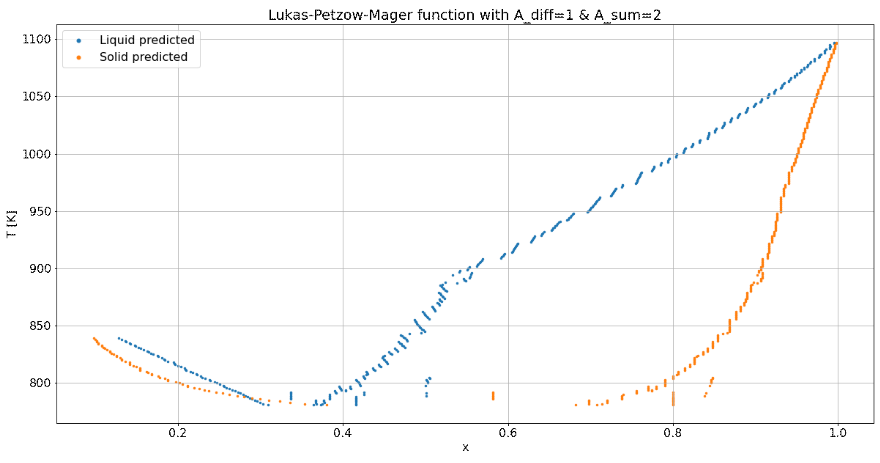


**Figure 6.31:** Predicted phase diagram for $A_{\text{diff}} = 1, A_{\text{sum}} = 2$

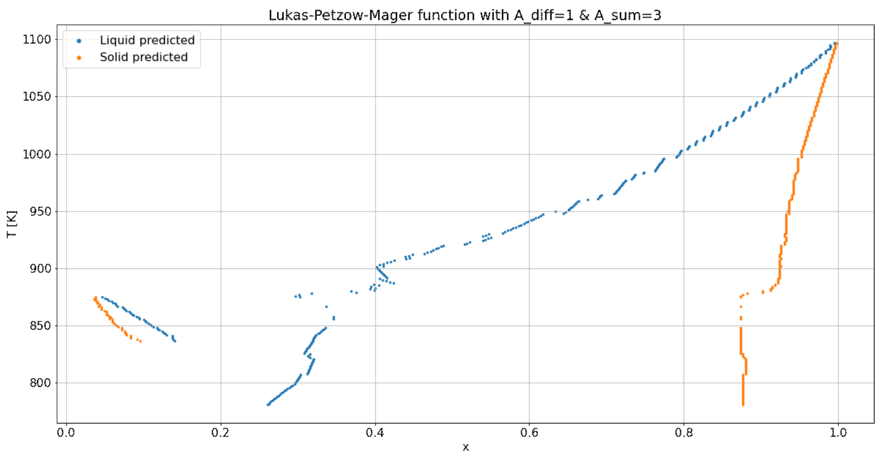


**Figure 6.32:** Predicted phase diagram for $A_{\text{diff}} = 1, A_{\text{sum}} = 3$

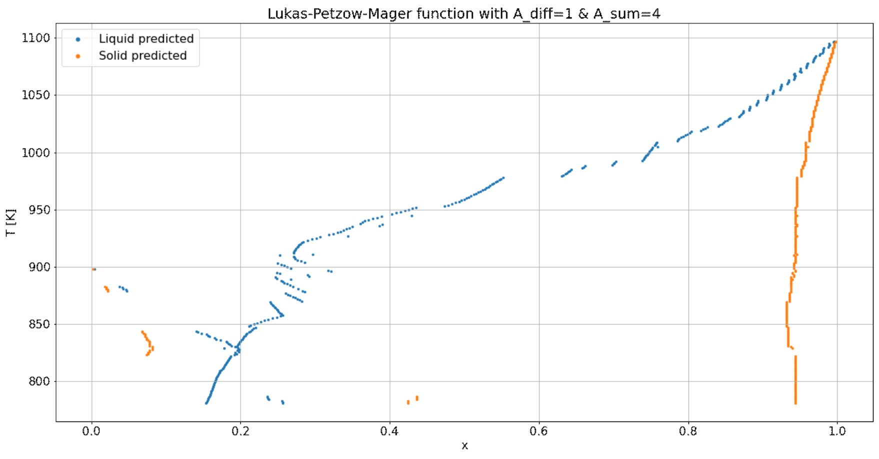


**Figure 6.33:** Predicted phase diagram for $A_{\text{diff}} = 1, A_{\text{sum}} = 4$

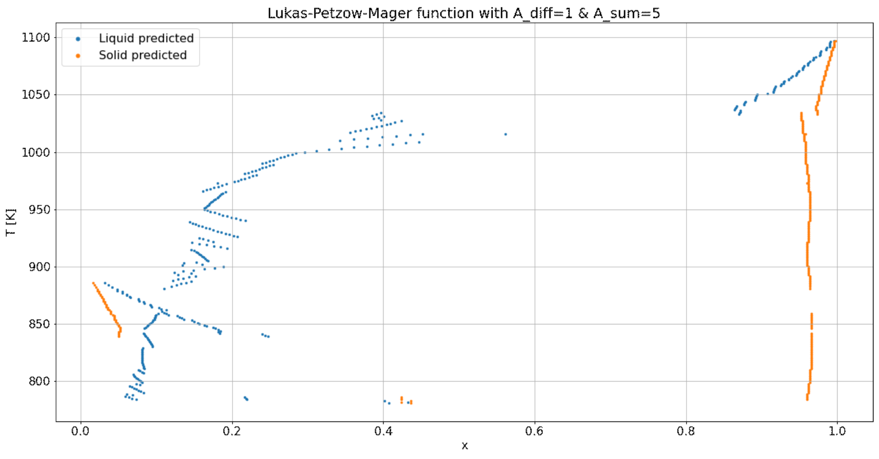


**Figure 6.34:** Predicted phase diagram for $A_{\text{diff}} = 1, A_{\text{sum}} = 5$

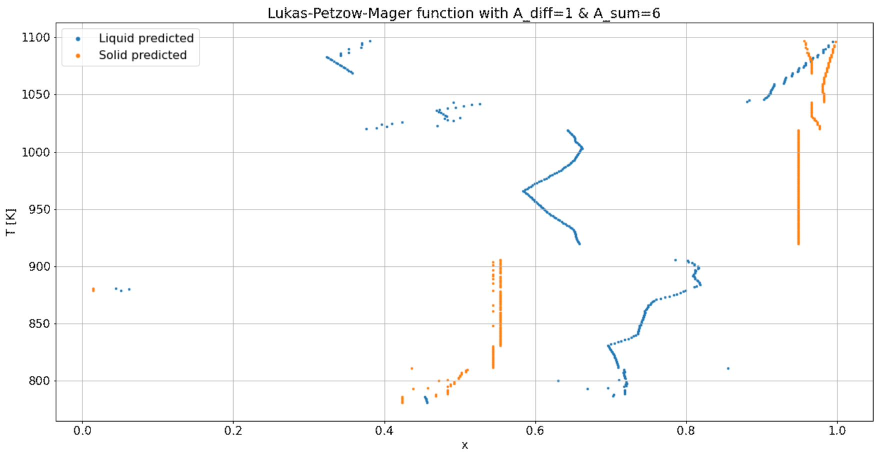


**Figure 6.35:** Predicted phase diagram for $A_{\text{diff}} = 1, A_{\text{sum}} = 6$

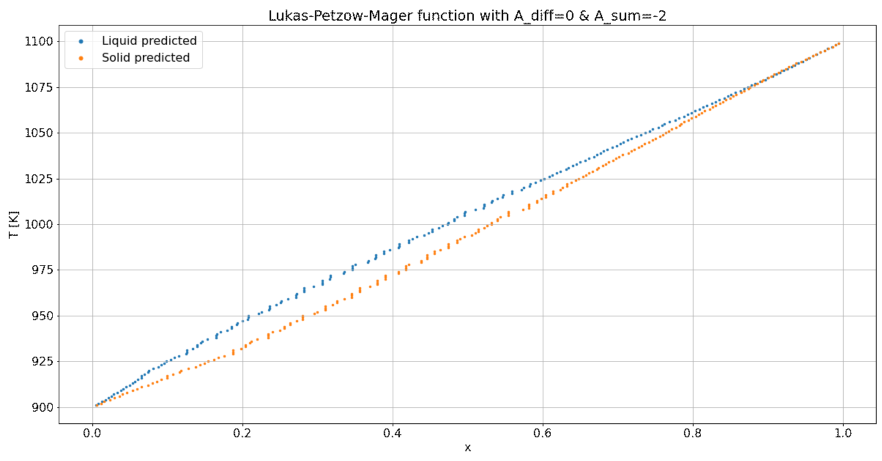


**Figure 6.36:** Predicted phase diagram for $A_{\text{diff}} = 0, A_{\text{sum}} = -2$

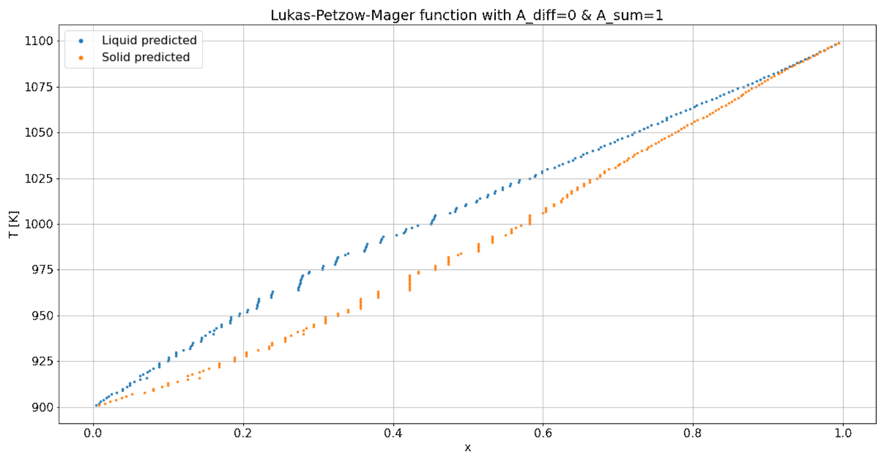


**Figure 6.37:** Predicted phase diagram for $A_{\text{diff}} = 0, A_{\text{sum}} = 1$

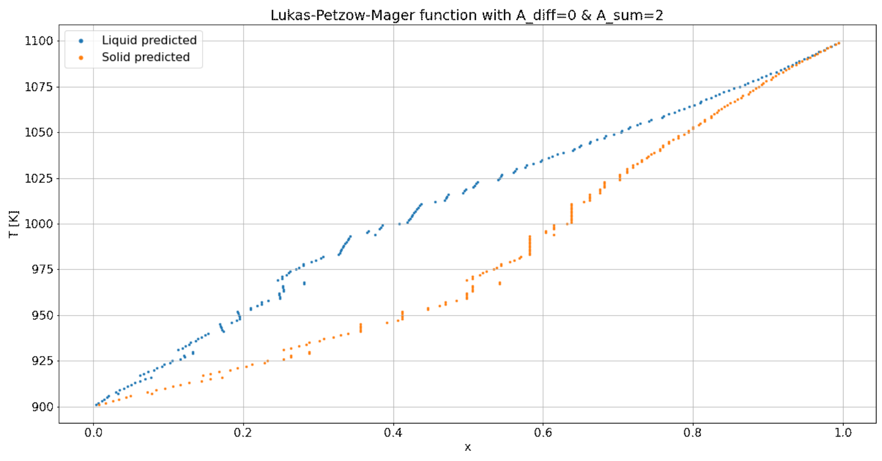


**Figure 6.38:** Predicted phase diagram for $A_{\text{diff}} = 0, A_{\text{sum}} = 2$

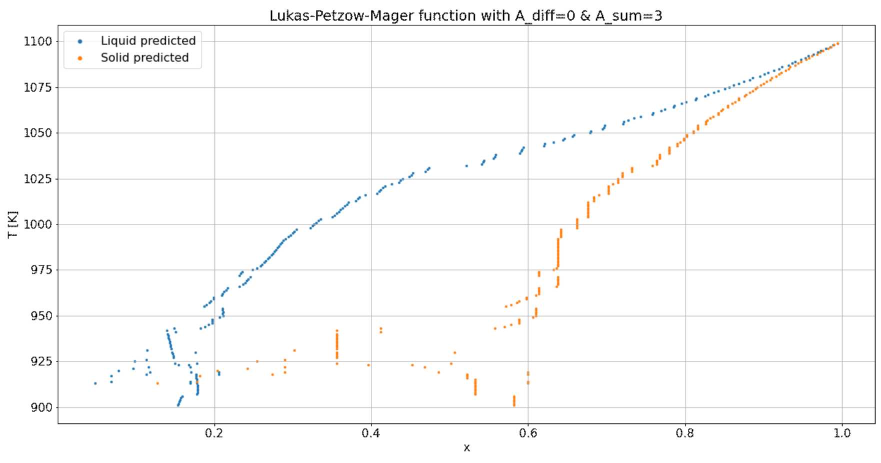


**Figure 6.39:** Predicted phase diagram for $A_{\text{diff}} = 0, A_{\text{sum}} = 3$

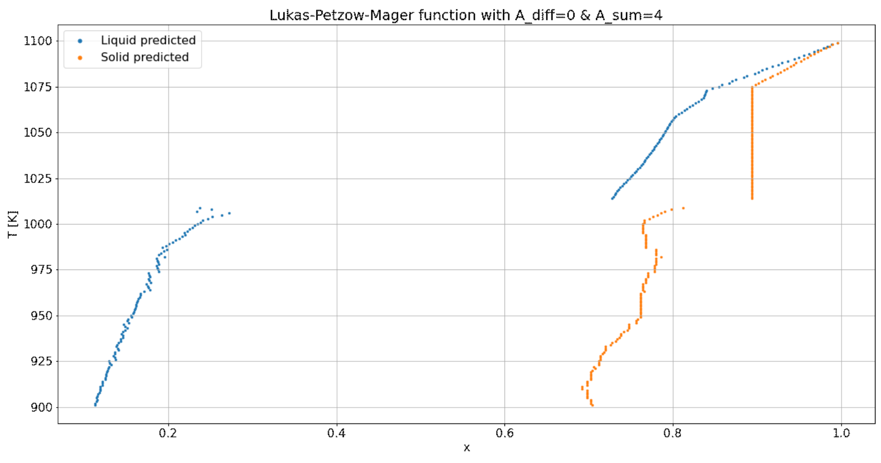


**Figure 6.40:** Predicted phase diagram for $A_{\text{diff}} = 0, A_{\text{sum}} = 4$

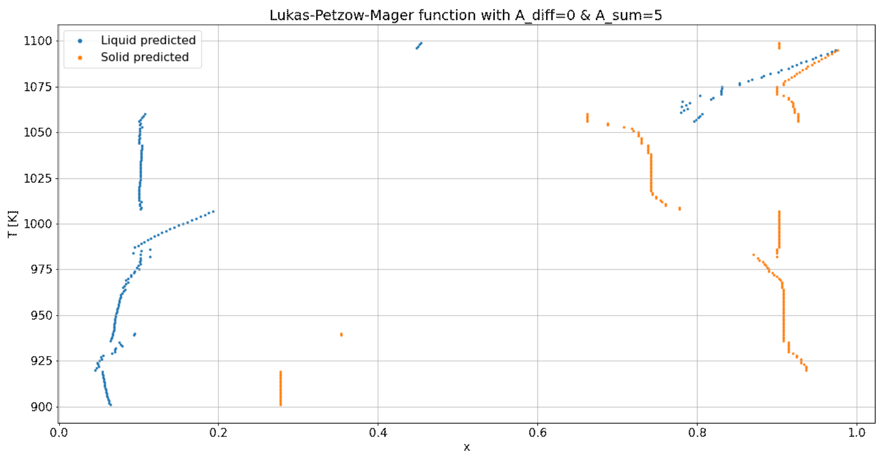


**Figure 6.41:** Predicted phase diagram for $A_{\text{diff}} = 0, A_{\text{sum}} = 5$

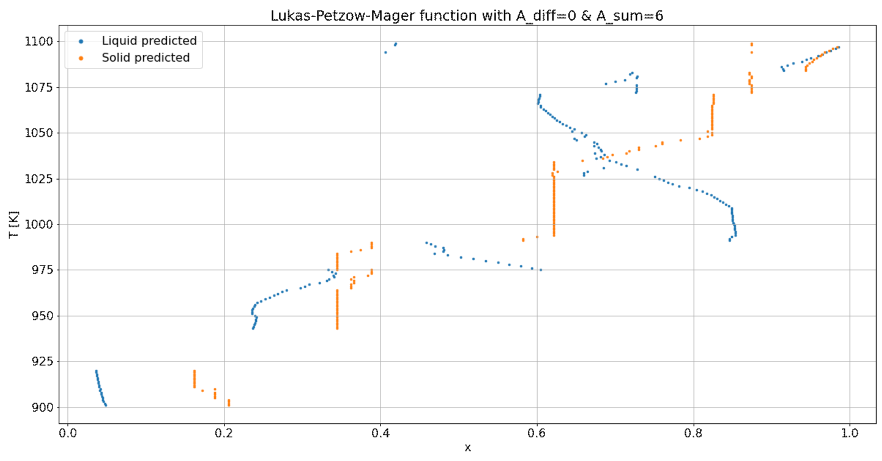


**Figure 6.42:** Predicted phase diagram for $A_{\text{diff}} = 0, A_{\text{sum}} = 6$

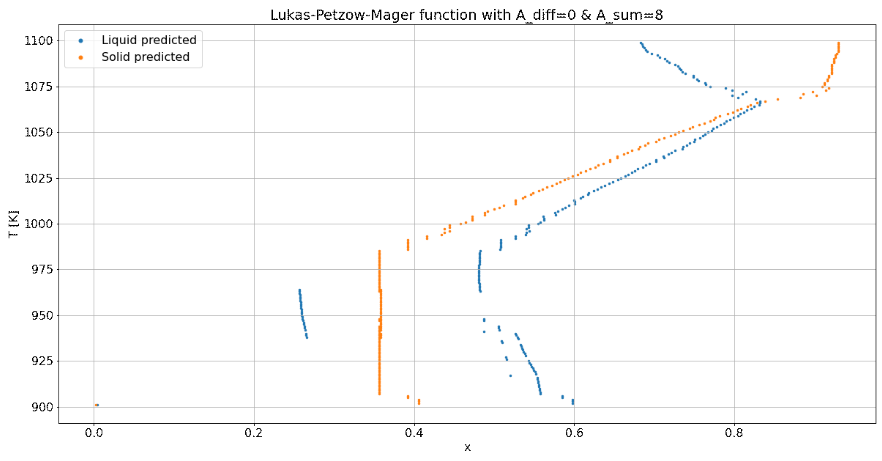


**Figure 6.43:** Predicted phase diagram for $A_{\text{diff}} = 0, A_{\text{sum}} = 8$

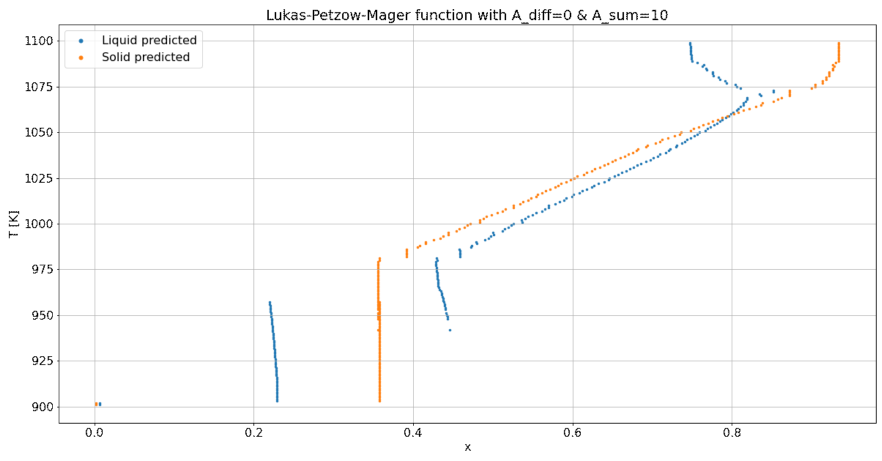


**Figure 6.44:** Predicted phase diagram for $A_{\text{diff}} = 0, A_{\text{sum}} = 10$

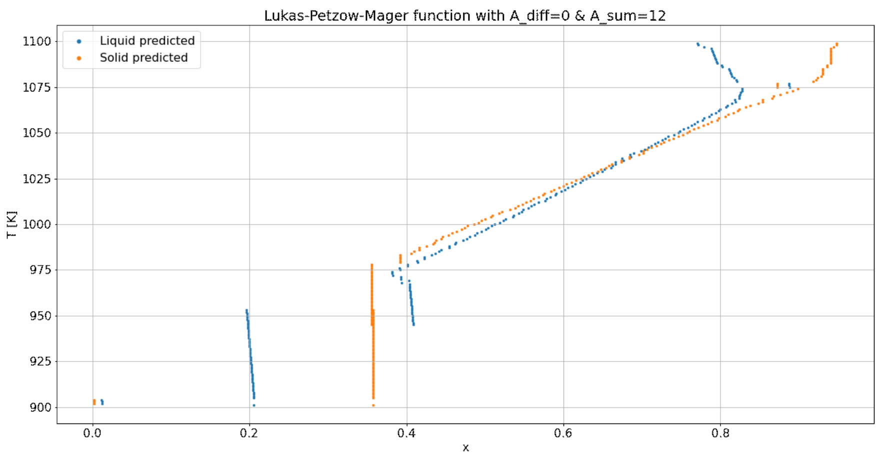


**Figure 6.45:** Predicted phase diagram for $A_{\text{diff}} = 0, A_{\text{sum}} = 12$

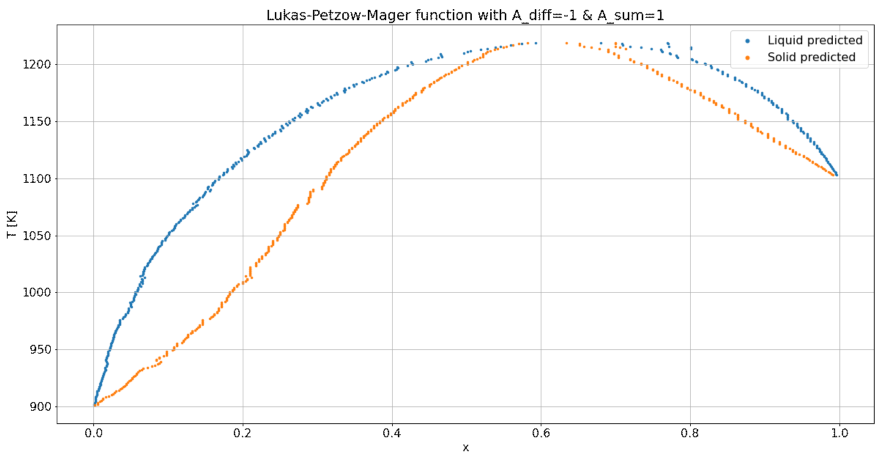


**Figure 6.46:** Predicted phase diagram for $A_{\text{diff}} = -1, A_{\text{sum}} = 1$

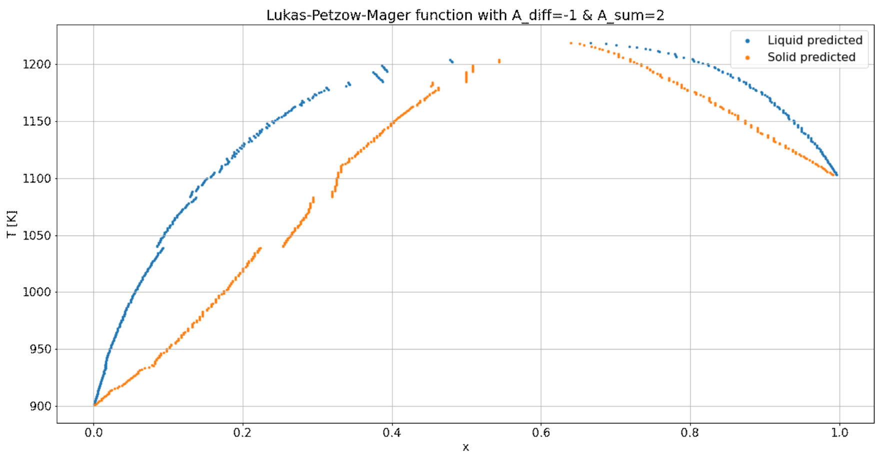


**Figure 6.47:** Predicted phase diagram for $A_{\text{diff}} = -1, A_{\text{sum}} = 2$

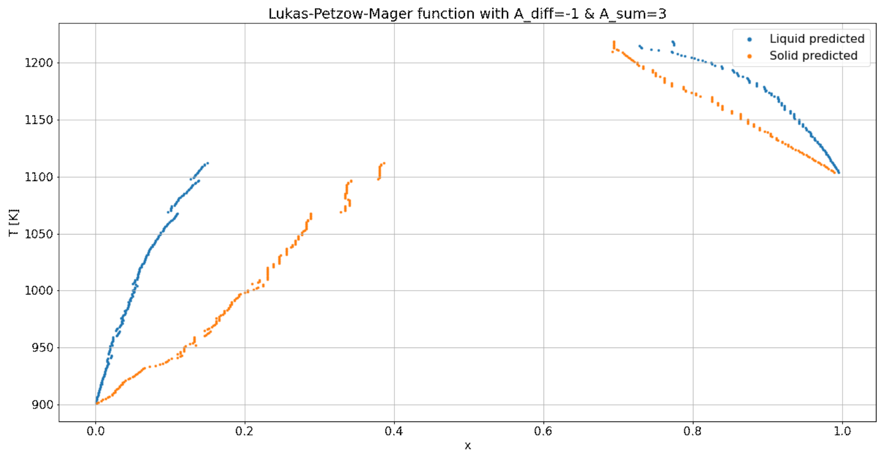


**Figure 6.48:** Predicted phase diagram for $A_{\text{diff}} = -1, A_{\text{sum}} = 3$

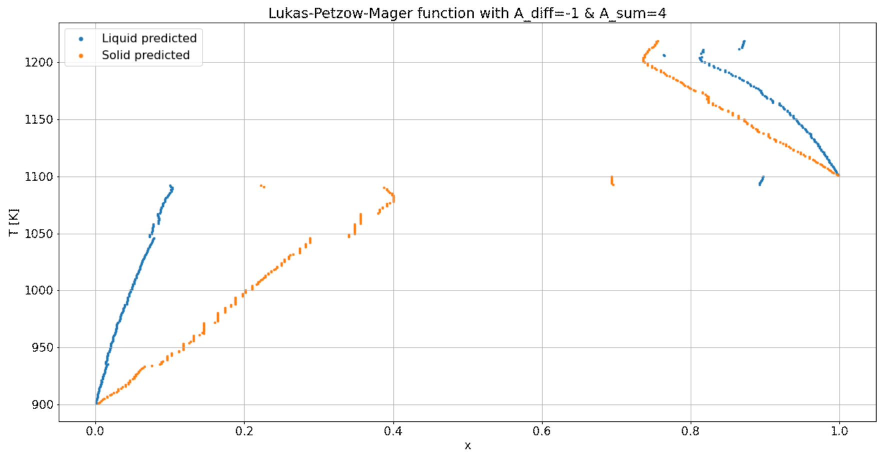


**Figure 6.49:** Predicted phase diagram for $A_{\text{diff}} = -1, A_{\text{sum}} = 4$

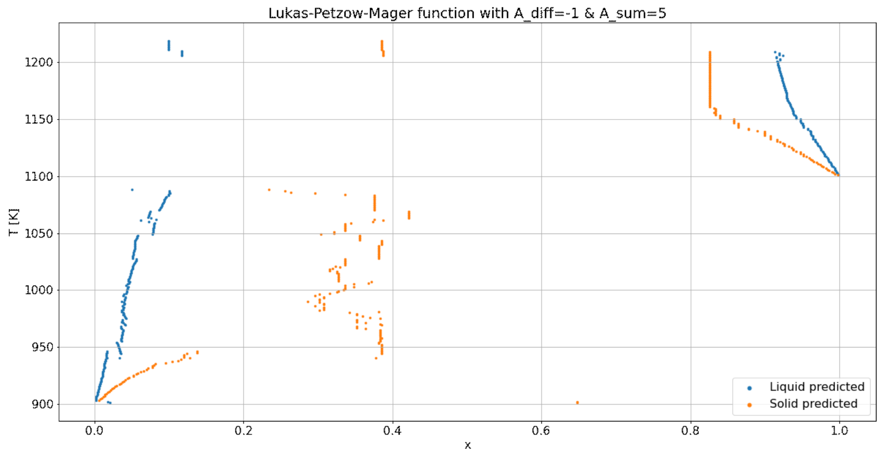


**Figure 6.50:** Predicted phase diagram for $A_{\text{diff}} = -1, A_{\text{sum}} = 5$

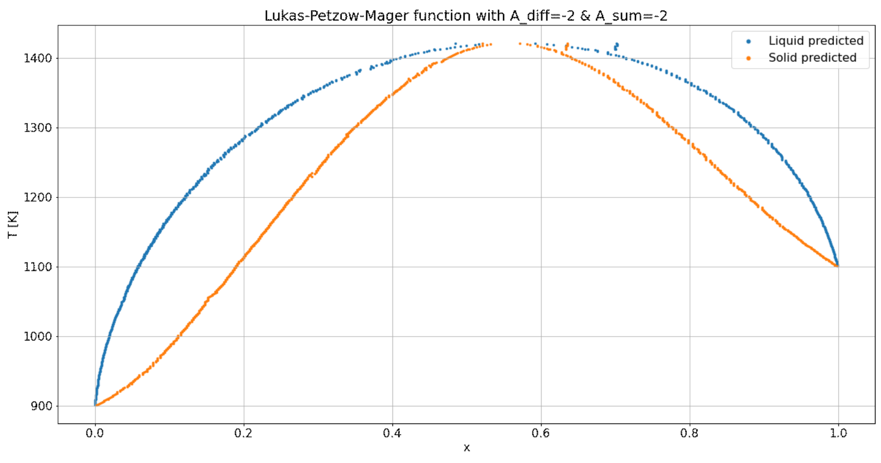


**Figure 6.51:** Predicted phase diagram for $A_{\text{diff}} = -2, A_{\text{sum}} = -2$

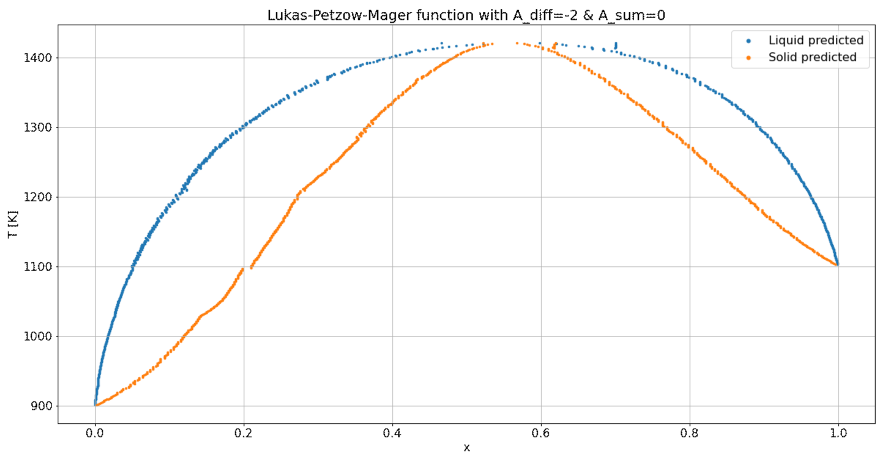


**Figure 6.52:** Predicted phase diagram for $A_{\text{diff}} = -2, A_{\text{sum}} = 0$

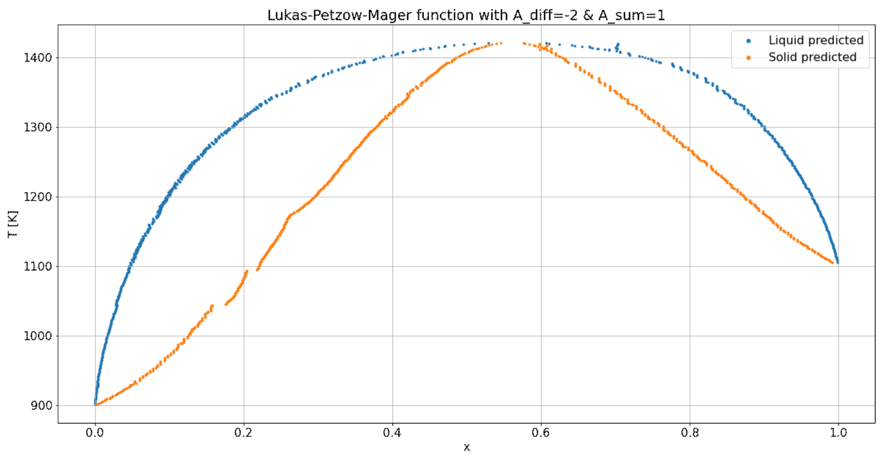


**Figure 6.53:** Predicted phase diagram for $A_{\text{diff}} = -2, A_{\text{sum}} = 1$

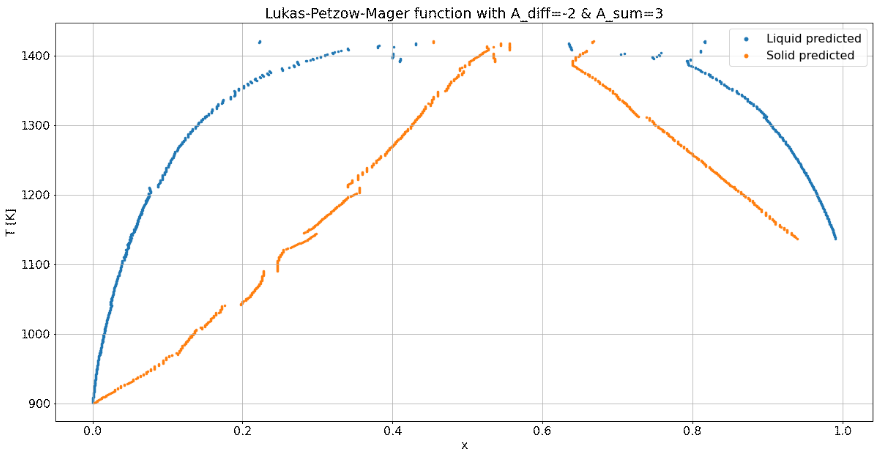


**Figure 6.54:** Predicted phase diagram for $A_{\text{diff}} = -2, A_{\text{sum}} = 3$

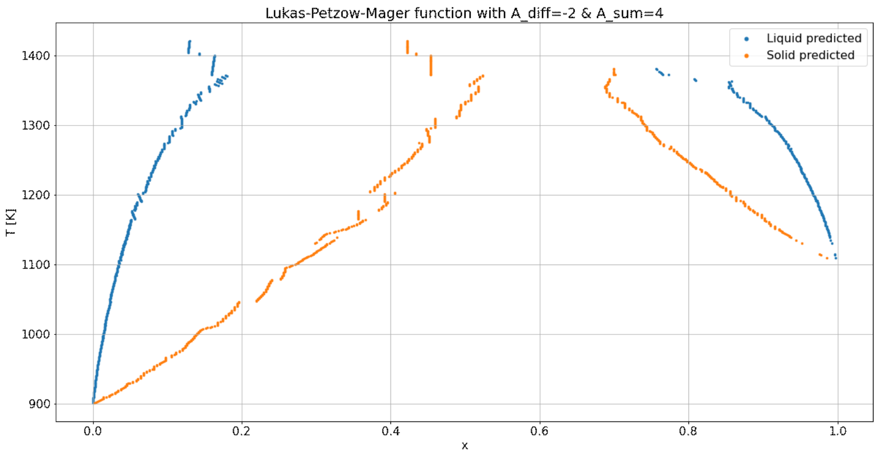


**Figure 6.55:** Predicted phase diagram for $A_{\text{diff}} = -2, A_{\text{sum}} = 4$

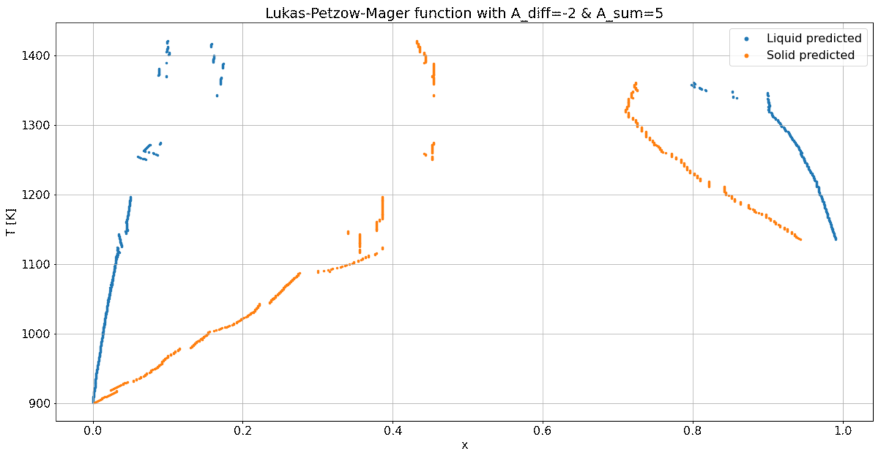


**Figure 6.56:** Predicted phase diagram for $A_{\text{diff}} = -2, A_{\text{sum}} = 5$

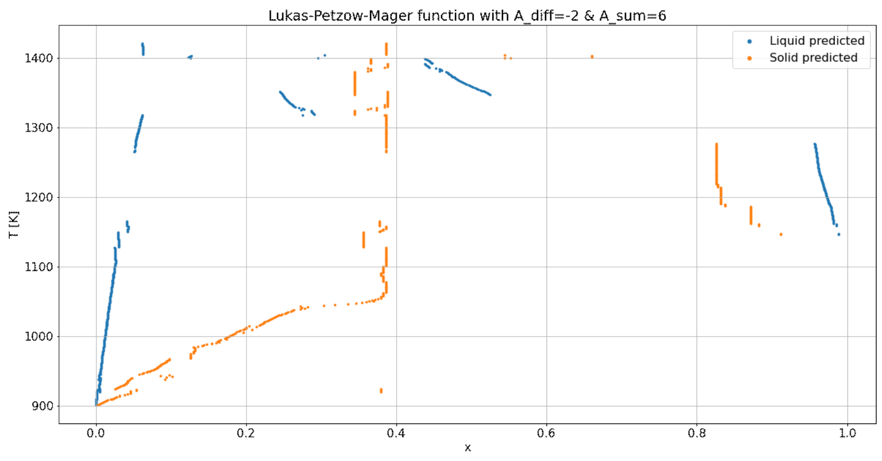


**Figure 6.57:** Predicted phase diagram for $A_{\mathrm{diff}} = -2, A_{\mathrm{sum}} = 6$

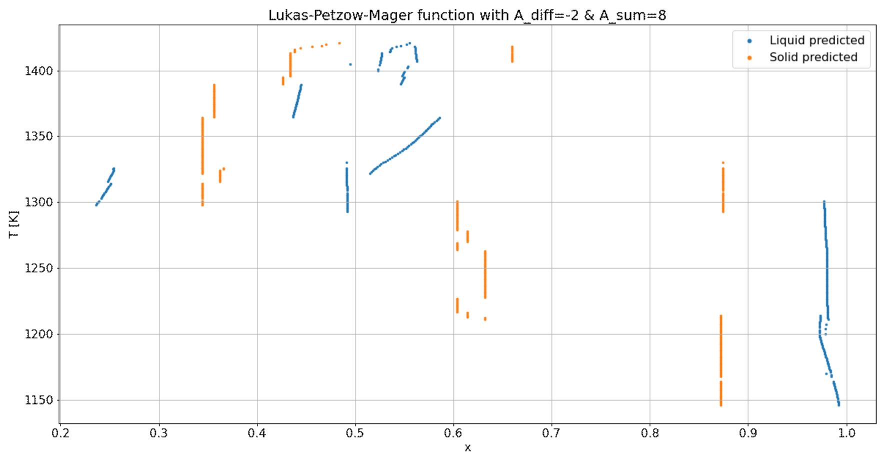


**Figure 6.58:** Predicted phase diagram for $A_{\mathrm{diff}} = -2, A_{\mathrm{sum}} = 8$

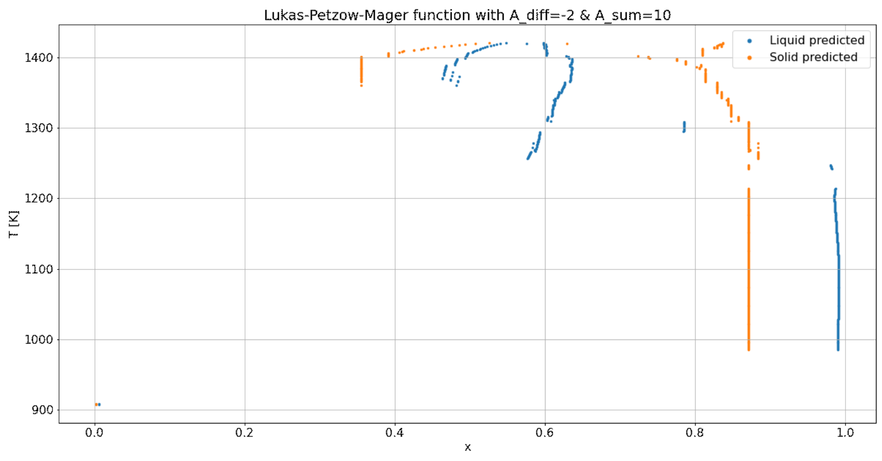


**Figure 6.59:** Predicted phase diagram for $A_{\mathrm{diff}} = -2, A_{\mathrm{sum}} = 10$

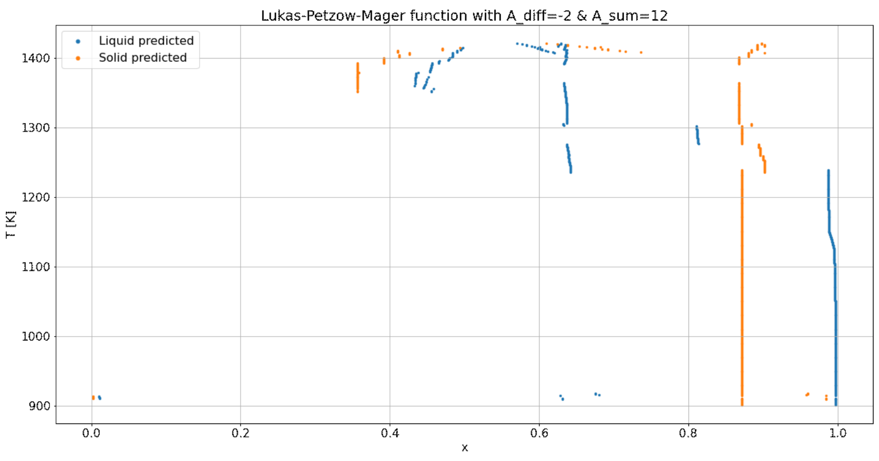


**Figure 6.60:** Predicted phase diagram for $A_{\text{diff}} = -2, A_{\text{sum}} = 12$

## 6.2 Code

All the code written for this thesis is available at https://github.com/eldonko/Masters-Thesis-Code

# 7 Literatur

[1] Springboard: Artificial Intelligence Future – Less Artificial, More Intelligent. URL: https://www.springboard.com/blog/data-science/artificial-intelligence-future/#:~:text=According%20to%20a%20report%20on,it%20will%20take%20away%20jobs. Abrufdatum 04.05.22 09:21.

[2] Contract Pharma: How Artificial Intelligence and Machine Learning are Transforming the Life Sciences. URL: https://www.contractpharma.com/contents/view_experts-opinion/2022-01-25/how-artificial-intelligence-and-machine-learning-are-transforming-the-life-sciences/. Abrufdatum 04.05.22 09:18.

[3] Silver, D.; Schrittwieser, J.; Simonyan, K.: Mastering the Game of Go without Human Knowledge. URL: https://discovery.ucl.ac.uk/id/eprint/10045895/1/agz_unformatted_nature.pdf. Abrufdatum 22.02.2022.

[4] Länge, M.: An artificial neural network model for the unary description of pure substances and its application on the thermodynamic modelling of pure iron. In: Soft Computing 24 (2020) 16, S. 12227–39.

[5] Kan, P.; Lee, C.-J.: A Neural Network Model for Prediction of Phase Equilibria in Aqueous Two-Phase Extraction [Kan, P.; Lee, C-J.]. In: Ind. Eng. Chem. (1996), S. 2015–23.

[6] Bilgin, M.; Hasdemir, M.; Otas, O.: The use of neural networks on VLE data prediction. In: Journal of Scientific & Industrial Research (2004) 63, S. 336–43.

[7] Farzi, A.; Tarjoman Nejad, A.: Prediction of phase equilibria in binary systems containing acetone using artificial neural network. In: International Journal of Scientific & Engineering Research, (2015), S. 358–63.

[8] Kondepudi, D.; Prigogine, I.: Modern Thermodynamics. From Heat Engines to Dissipative Structures. Chichester 1999.

[9] Planck, M.: Treatise on Thermodynamics, 3. Auflage. New York: Dover 1945.

[10] Baehr, H. D.: Thermodynamik. Grundlagen und technische Anwendungen, 15. Auflage 2012.

[11] Moran, M. J.; Shapiro, H. N.; Boettner, D. D.; Bailey, M. B.: Fundamentals of Engineering Thermodynamics, 8. Auflage. Hoboken, New Jersey, USA 2014.

[12] Mager, T.; Lukas, H. L.; Petzow, G.: Statistische Konstitutionsanalyse - Thermodynamische Konstitutionsmorphologie [Mager, T.; Lukas, H. L.; Petzow, G.]. In: Z. Metallkde. (1972) 63, S. 638–47.

[13] Bale, C. W.; Bélisle, E.; Chartrand, P.; Decterov, S. A.; Eriksson, G.; Gheribi, A. E.; Hack, K.; Jung, I.-H.; Kang, Y.-B.; Melançon, J.; Pelton, A. D.; Petersen, S.; Robelin, C.; Sangster, J.; Spencer, P.; van Ende, M.-A.: FactSage thermochemical software and databases, 2010–2016. In: Calphad 54 (2016), S. 35–53.

[14] Dinsdale, A. T.: SGTE data for pure elements.

[15] Barin, I.: Thermochemical Data of Pure Substances, 3. Auflage. D-69451 Weinheim (Federal Republic of Germany) 1995.

[16] ThermoCalc. URL: https://thermocalc.com/.

[17] Smola, A.; Vishwanathan, S. V. N.: Introduction to Machine Learning. Cambridge, United Kingdom 2008.

[18] Mitchell, T. M.: Key Ideas in Machine Learning. URL: https://www.cs.cmu.edu/~tom/mlbook/keyIdeas.pdf. Abrufdatum 22.02.2022.

[19] Jung, A.: Machine Learning. Singapore 2022.

[20] Bishop, C. M.: Pattern Recognition and Machine Learning. Singapore 2006.

[21] Gurney, K.: An Introduction to Neural Networks. Hoboken 2003.

[22] Baheti, P.: 12 Types of Neural Network Activation Functions: How to Choose? URL: https://www.v7labs.com/blog/neural-networks-activation-functions. Abrufdatum 23.02.2022.

[23] Nielsen, M.: Neural Networks and Deep Learning. URL: https://static.latexstudio.net/article/2018/0912/neuralnetworksanddeeplearning.pdf. Abrufdatum 23.02.2022.

[24] Stanford University: Neural Nets and Deep Learning. URL: http://infolab.stanford.edu/~ullman/mmds/ch13.pdf. Abrufdatum 23.02.2022.

[25] Tensorflow. URL: https://www.tensorflow.org/.

[26] Pytorch. URL: https://pytorch.org/.

[27] Lashkarbolooki, M.; Shafipour, Z. S.; Hezave, A. Z.; Farmani, H.: Use of artificial neural networks for prediction of phase equilibria in the binary system containing carbon dioxide. In: The Journal of Supercritical Fluids 75 (2013), S. 144–51.

[28] Chen, Q.; Sundman, B.: Modeling of thermodynamic properties for Bcc, Fcc, liquid, and amorphous iron. In: Journal of Phase Equilibria 22 (2001) 6, S. 631–44.

[29] Chemistry LibreTexts. URL: https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Physical_Chemistry_(LibreTexts)/17%3A_Boltzmann_Factor_and_Partition_Functions/17.04%3A_Heat_Capacity_at_Constant_Volume. Abrufdatum 02.05.22 12:35.

[30] Guinier, A.; Jullien, R.: Die physikalischen Eigenschaften von Festkörpern - Eine leichtverständliche Einführung in die Festkörperphysik [Guinier, A.; Jullien, R.].

[31] Wikipedia: LogSumExp. URL: https://en.wikipedia.org/wiki/LogSumExp. Abrufdatum 02.05.22 13:25.

[32] Hassam, S.; Gambino, M.; Gaune-Escard, M.; Bros, J. P.; Agren, J.: Experimental and Calculated Ag + Au + Ge Phase Diagram. In: METALLURGICAL TRANSACTIONS A (1988) 19A.

[33] Kingma, D. P.; Ba, J.: Adam: A Method for Stochastic Optimization.