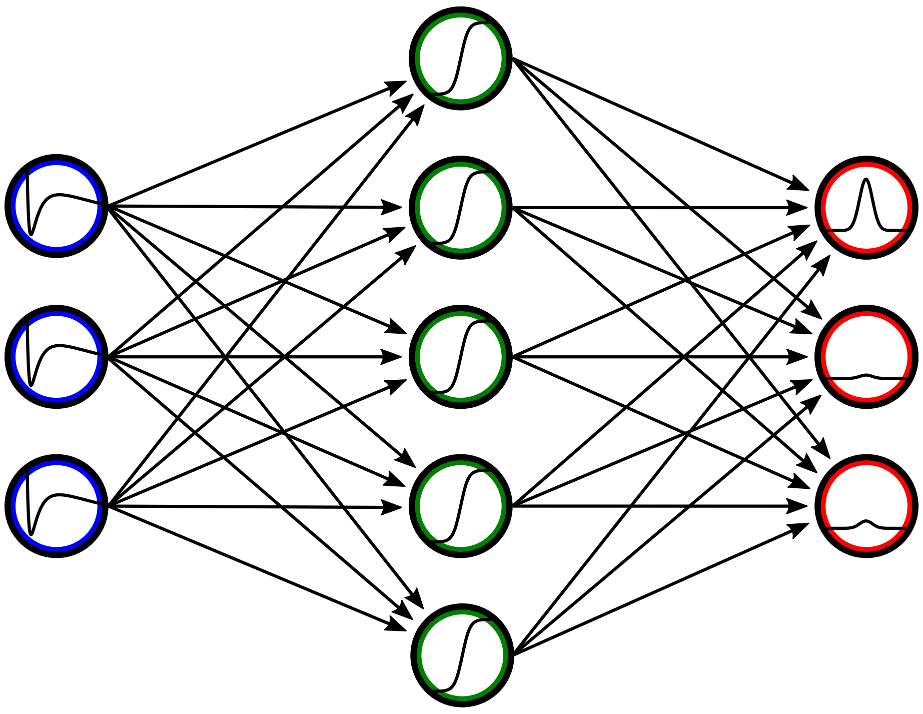
Solving Vapor-Liquid Flash Problems Using Artificial Neural Networks

Jonah Pieter Poort



Solving Vapor-Liquid Flash Problems Using Artificial Neural Networks

by

Jonah Pieter Poort

in partial fulfillment of the requirements for the degree of

Master of Science
in Mechanical Engineering

at the Delft University of Technology,
to be defended publicly on Tuesday October 30, 2018 at 13:30 PM.

Student number:	4230116	
Project duration:	January 2018 – October 2018	
Daily supervisor:	Ir. J. van Kranendonk,	ZEF B.V.
Thesis committee:	Prof.dr.ir. T.J.H. Vlugt,	TU Delft, P&E (ETh) | Chair
	Dr.ir. M. Ramdin,	TU Delft, P&E (ETh)
	Prof.dr.ir. W. de Jong,	TU Delft, P&E (LSE)
	Dr.ir. R. Delfos,	TU Delft, P&E (FM)

Abstract

Vapor-liquid phase equilibrium (flash) calculations largely contribute to the total computation time of many process simulation models. As a result, process simulations, especially dynamic cases, are limited in the amount of detail that can be included due to time restrictions. In addition, under certain conditions flash calculations can fail to provide acceptable results. In this work, artificial neural networks were investigated as a potentially faster and more robust alternative to conventional flash calculation methods. Classification neural networks were used to determine the phase stability of a given mixture of fluids, while regression networks were used to make predictions of thermodynamic property values. In addition to conventional flash types such as the constant pressure, constant temperature (PT), and constant pressure, constant entropy (PS) flash, neural networks are used to develop two concept flash types: a constant entropy, constant volume (SV), and a constant enthalpy, constant volume (HV) flash. All neural networks were trained on, and compared to, data generated using the PT-flash algorithm from the Thermodynamics for Engineering Applications (TEA) property calculator. Data was generated for mixtures of water and methanol over a wide range of pressures and temperatures. The artificial neural networks showed speed improvements over TEA of up to 35 times for phase classification and 15 times for property predictions. Overall phase classification accuracy scores of around 97% were achieved. Average property value prediction errors range between 0.5% and 7% when compared to the spread in magnitude of the test data, and R^2 scores were in the general order of 0.95 and higher, although classification and property prediction in the two-phase region showed markedly higher errors than properties in the pure liquid or vapor regions. Moreover, thermodynamic consistency and the stability of a system consisting of multiple neural network flash types both still require considerable improvement. Finally, this work shows that artificial neural networks can be used to create unconventional flash types such a the SV- and HV-flash.

Acknowledgements

First and foremost, I would like to thank Jan van Kranendonk for coming up with the idea for this project, providing me with valuable feedback throughout its development, and having been, for most of the duration of this project, the only person who understood anything of what I was doing.

I would also like to thank my supervisors Thijs Vlugt and Mahinder Ramdin for the advice they gave me, and their help in turning this project into something worthy of publishing in a scientific journal.

Furthermore, my gratitude goes to the people at ZEF for getting coffee with me whenever I was waiting for data to generate or neural networks to be trained. Special thanks to Hessel and Ulrich for their help, guidance, and wise words of advice on things less technical. Thanks to my father, Pratik Basarkar, and Anshul Garg for proof-reading my report and providing feedback.

Finally, I would like to thank my parents, sister, and friends for their support throughout.

Jonah Pieter Poort,
Delft, October 2018.

Contents

List of Figures

List of Tables

Nomenclature

Roman symbols

A	Helmholtz free energy	[J/mol]
a	EOS parameter	[Jm^3/mol^2]
b	EOS parameter Bias term	[m^3/mol]
C_p	Heat capacity	[J/mol/K]
c	Number of components in mixture	
F	Number of degrees of freedom	
f	Fugacity	[Pa]
G	Gibbs free energy	[J/mol]
H	Enthalpy	[J/mol]
J	Loss function value	
K	Equilibrium constant	
k	EOS parameter	
N	Total mole number	[mol]
n	Component mole number Number of entries in set	[mol]
o	Predicted output value	
P	Pressure	[Pa]
R	Universal gas constant	8.134 J/mol/K
R^2	Coefficient of deviation	
S	Entropy Total signal	[J/mol/K]
s	Individual signal	
T	Temperature	[K]
t	Target value	
U	Internal energy	[J/mol]
V	Volume	[m^3/mol]
w	Weight term	

x	Liquid mole fraction General function variable	[mol/mol]
y	Vapor mole fraction	[mol/mol]
Z	Compressibility factor	
z	Total/feed mole fraction	[mol/mol]
$\boldsymbol{A}$	Activation vector	
$\boldsymbol{B}$	Bias vector	
$\boldsymbol{S}$	Signal vector	
$\boldsymbol{W}$	Weight matrix	

Greek symbols

β	Vapor fraction	[mol/mol]
ϵ	Error term	
η	Viscosity	[Pa·s]
λ	Thermal conductivity	[W/m/K]
μ	Chemical potential Mean value	[J/mol]
π	Number of equilibrium phases	
σ	Standard deviation	
ϕ	Fugacity coefficient	
ω	Acentric factor	

Subscripts

a	Activation
c	Critical value
cor	Correct value
i	Index of entry in set
id	Ideal gas value
j	Index of entry in set
mix	Mixture value
m	Methanol
p	Propagating layer
r	Receiving layer Reduced value
ref	Reference state
sc	Scaled value

test	Test data set
w	Water

Superscripts

F	Feed
k	Iteration number
L	Liquid phase
V	Vapor phase

Abbreviations

ANN	Artificial Neural Network
APE	Absolute Percentage Error
BA	Binary Accuracy
CAPE	Computer-Aided Property Estimation
CE	Cross-Entropy
CPU	Central Processing Unit
COCO	CAPE-OPEN to CAPE-OPEN
ECM	Error Contribution of Misclassifications
EOS	Equation of State
FNN	Feedforward Neural Network
GA	Genetic Algorithm
GPU	Graphics Processing Unit
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SLDR	Scaling Law of Rectilinear Diameter
STLL	Stability Test Limit Locus
TEA	Thermodynamics for Engineering Applications
UAT	Universal Approximation Theorem

Chapter 1

Introduction

1.1 Project context

Process simulation is an indispensable tool in the design of process equipment, and investigating the viability of new, or optimizing the performance of current, process designs [1]. Through the mathematical modelling of phenomena such as heat transfer, chemical reactions, and phase splitting, process simulations make it possible to predict the effects design choices and operating conditions will have on the performance of a future plant. In this way, the need for a physical representation of the plant can be avoided during early design stages, significantly saving on cost and time.

At the basis of any process simulation model lie so-called *fluid property calculations* [2], which, as the name implies, calculate physical properties of the (mixtures of) fluids present in the plant. Fluid property calculations are not only crucial in determining material properties required in order to accurately model phenomena such as reactions and heat and mass transfer, they are essential to the proper modelling of separation processes such as distillation, absorption, and stripping.

Fluid properties include, among others, thermodynamic properties such as pressure (P), volume (V), and temperature (T), and transport properties such as viscosity (η), and thermal conductivity (λ). They are in most cases determined through the use of a *flash* algorithm, which calculates the values of all thermodynamic properties of a given fluid based on specified values of only two of them [1, 3]. The flash algorithm also determines the fluids *state of aggregation* (phase), i.e. whether it is liquid, vapor, or in a state of equilibrium between the two. If the fluid in question is a mixture of multiple chemical compounds, in addition to the two thermodynamic properties the (relative) amounts of each mixture component must also be specified.

As knowledge of fluid property values is a prerequisite to all other process simulation calculations, they naturally take up a major part of the total time it takes to execute a single simulation, up to 50-70% in certain cases [4, 5]. For *steady-state* simulations, which assume that inputs and outputs of the plant are constant, simulation times are often not much of a consideration, as most calculations will have to be performed only a limited number of times.

On the other hand, an increasing number of process simulation applications require the use of dynamic models, either because of the need for more accurate simulation results, or the fact that the process at hand is dynamic in nature and cannot be modelled as steady-state [6]. Dynamic simulation models can provide more detailed and better understanding of many processes, but are much more complex than steady-state models, having to take into account the state of the system at a previous point in time in order to be able to predict the state at a future point in time. As a result, all calculations will have to be repeated a large number of times and simulation times, to which flash calculation contribute a majority share, can become a severely limiting factor. In addition to slow executions times, conventional flash methods can also encounter situations in which they fail to provide an answer, especially near the critical point and phase boundaries [7,8].

There is therefore a need for a robust computational method which can improve on the computational burden of current flash calculation approaches, paving the way for a more widespread application of dynamic simulations in process design.

One group of computational methods that has recently attracted interest for a multitude of applications, are Artificial Neural Networks (ANN). ANN are a broad collection of computational methods inspired by the biological networks of neurons found in animal brains. ANN form a subset of the greater field of Machine Learning (ML), and are capable of learning complex non-linear relationships or structures within large sets of data through a process of repeated exposure and adaptation of the networks inner parameters to said data, referred to as training [9]. Due to the basic structure of ANN, they rarely fail to provide satisfactory results (if properly trained), and are very fast to execute [5]. Nowadays, ANN are applied in almost every field of study, from physics, biology, and engineering, to agriculture, sociology, and economics, and are used by most large tech-companies including Google, YouTube, Amazon, and Facebook.

In this thesis, ANN will be investigated as a potential alternative to conventional flash calculation algorithms due to their inherent robustness and fast execution speed.

1.1.1 Zero Emission Fuels

This project is done on behalf of, and in close collaboration with Zero Emission Fuels (ZEF / ZEF B.V.), a start-up company based in Delft, the Netherlands, working together with the Delft University of Technology, the Dutch research institute TNO, and surrounding technological institutes and companies in order to develop a highly integrated small-scale methanol production plant.

Each of ZEF's production plants is paired with a solar panel that can deliver up to 300 Watt. This energy is used to capture carbon dioxide and water out of air, split water into oxygen and hydrogen, and react the hydrogen along with the carbon dioxide into methanol [10]. Methanol is an energy-dense alcohol-like substance that is liquid at room temperature and can be utilized directly as fuel for cars, ships, planes, or power generators, or as a building block in a multitude of chemical processes [10]. A schematic overview of the plant is shown in Figure 1.1.

By integrating the different subsystems of the typical synthetic methanol production chain, reducing their size, weight, and number of parts, and having each plant function autonomously, ZEF aim to significantly reduce the capital and operational costs of synthetic methanol production. Ultimately making it not only economically feasible, but lucrative.

Due to its small size and the intermittent nature of its power source, the ZEF methanol plant is expected at all times to function in a dynamic state. A dynamic simulation model is therefore needed in order to aid in the design of improvements to the plant. Due to the complexity of the plant, existing process simulation tools have been found to be lacking in their predictive capabilities, and the amount of time each simulation takes.

For this reason, a significant part of ZEF's time and resources have been and are being devoted to the development of an in-house dynamic simulation engine capable of simulating in great detail the behavior of the plant throughout an entire day of operation. Previous experiences have ultimately lead them to artificial neural network as a possible solution to the speed and robustness problems encountered in conventional methods.

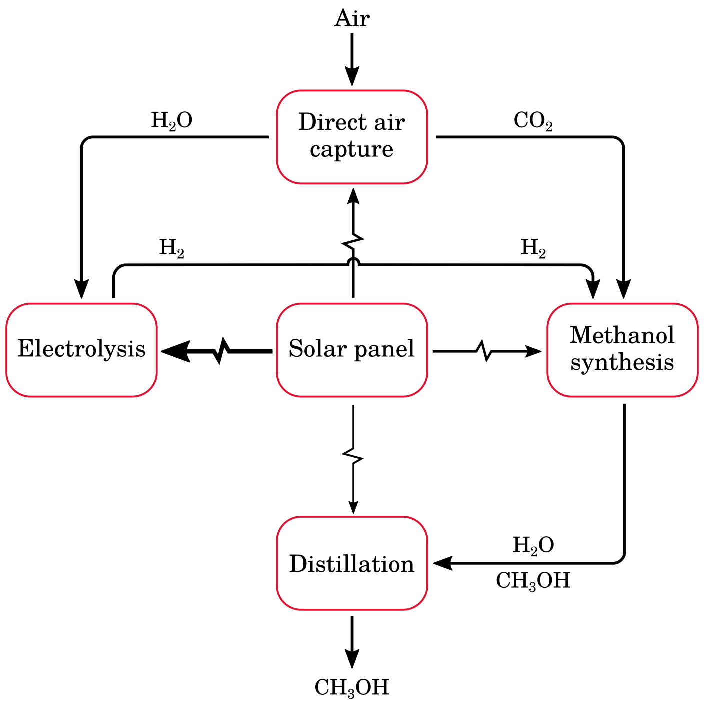


Figure 1.1: Schematic overview of the small-scale ZEF methanol plant [10]. A solar panel provides electricity to all subsystems of the plant; the direct air capture unit captures water and carbon dioxide out of air; the electrolysis cell splits the water into hydrogen and oxygen; the methanol synthesis reactor converts the hydrogen along with the carbon dioxide into methanol; and the micro-distillation unit separates the water-methanol mixture coming from the reactor into high purity methanol and water.

1.2 Research questions

The aim of this thesis is to investigate the potential application of artificial neural networks in augmenting current flash calculations in order to facilitate quicker computation times and improve robustness of the calculations. Based on this goal, a handful of research questions have been formed which this thesis will aim to answer:

1. In which way can you frame conventional flash problems in order for them to be solvable using artificial neural networks?
2. Can artificial neural networks be trained to solve one type of flash problem solely based on data obtained from another type of flash problem?
3. What speed improvement can be achieved through the use of artificial neural networks?
4. What is the resulting accuracy of the artificial neural networks?

5. Can an increase in robustness be achieved through the application of neural networks?

6. To what extent are the resulting neural network flashes thermodynamically consistent?

7. How stable is a system of artificial neural network flash algorithms?

Through answering the above questions, this thesis aims to give preliminary conclusions on the feasibility of the neural network approach to flash calculations, highlight areas which require further attention, and provide recommendations on strategies for future improvement.

1.3 Project goals

In order to answer the research questions posed in the previous section, this thesis has been divided into a number of goals:

1. Find an approach to solve the classical flash problem through the use of artificial neural networks.

2. Generate the appropriate data and train neural networks on the generated data.

3. Develop an implementation of the networks in the Python programming language such that they can be used for fluid property calculations.

4. Quantify the performance of the resulting artificial neural network flash calculations and identify points of improvements.

5. Provide conclusions and recommendations on current feasibility and areas of improvement.

1.4 Project boundaries

The topic of fluid phase equilibria is very broad, and the potential for applications of artificial neural networks is too large to be addressed in a single thesis. For this reason, a number of boundaries have been set in order to reduce the scope of this project to a more manageable size.

1.4.1 Phases

Only the fluid phases of vapor and liquid and their equilibria will be taken into account in this thesis. Not only do they form the majority of practical phase equilibrium problems [1], the vapor and liquid phase are the only phases expected to be present in the ZEF methanol plant.

1.4.2 Flash types

Flash calculations are typically specified by the thermodynamic variables that form the inputs to the calculation method. In theory there are dozens of flash calculation types that can be specified, but in practice only a select few have found widespread application. Based on the design of the in-house simulation tool being developed by ZEF B.V., the following flash types have been included in this thesis:

- A constant *pressure*, constant *temperature* flash.
- A constant *pressure*, constant *entropy* flash.
- A constant *enthalpy*, constant *volume* flash.
- A constant *entropy*, constant *volume* flash.

It should be noted that while the first two types mentioned above are used extensively in industry, the latter two methods have, to the authors knowledge, no clearly established framework which can be used to solve them using conventional methods. However, the versatility of ANN should make it possible to solve these two problems in exactly the same way as the first two, without the need for existing solution methods.

1.4.3 Chemical compounds

This thesis is restricted to include only the pure components of water and methanol, as well as mixtures of the two. Water and methanol have been chosen as they are the only two components expected to be present in the distillation system of the ZEF methanol plant, and thus form the simplest combination of chemical compounds which still have practical interest for ZEF B.V.

1.4.4 Data

All fluid property data used to train the ANN in this project have been generated using the Python plugin of the free-of-charge Thermodynamics for Engineering Applications (TEA) property calculator of the CAPE-OPEN to CAPE-OPEN (COCO) process simulator [11, 12].

Data was generated through conventional calculation methods using the Peng-Robinson equation of state as it can easily provide large sets of data on properties desired by ZEF for mixtures of components which might have limited availability in literature.

The COCO simulator was chosen over other process simulation or fluid property software alternatives such as Aspen Plus® or REFPROP as it is free-of-charge, which was important to ZEF B.V. due to licensing considerations.

However, the methods developed in this project are independent of the source of the training data, and as long as a large enough data set is available to achieve a desired accuracy, any valid data source will do.

1.4.5 Result optimization

A big risk in any large project is that of not knowing when to stop improving the results. There is always another approach one might try, another thing one could test, etc. In order to prevent this problem, this project will not include the optimization of speed, stability, or accuracy of its calculation results. This project is primarily focused on investigating the possibility and developing a methodology of applying artificial neural networks to phase equilibrium problems and quantifying its future potential.

1.5 Report structure

The theory relevant to this project is given in two chapters, Chapter 2 starts of with a description of the thermodynamics of phase equilibria, while Chapter 3 covers relevant topics in the field of artificial neural networks.

Next, Chapter 4 describes the development of the neural network flashes. Section 4.1 will outline the methods and implementation developed for solving the classic flash problem using neural networks, while Section 4.2 covers the topic of data generation and the process of training the neural networks.

Chapter 5 will provide the results of tests done to quantify the speed, accuracy, robustness, consistency, and stability of the developed neural network flash algorithm. Finally, Chapter 6 will give conclusions of the current work, and provide recommendations for future work.

Appendices at the end of this report will contain material found not be be within the direct scope of this report, but which could provide additional detail on certain subjects covered in the main body of this text.

Interlude: the language analogy

In order to clarify the reasoning behind the application of neural networks to the flash problem, an analogy can be made to learning a new language. Learning a completely new language is a difficult and time-consuming endeavor; when going on vacation to a foreign country for only a few days or weeks, it is generally not worth the time and effort to learn the local language. It will most likely suffice to look up some phrases here and there. On the other hand, if you permanently move abroad, learning the language will very likely pay off in the long run. While it is theoretically possible to look it up in the dictionary every time you come across a word you don't understand, the frequency at which you'll encounter new words and phrases is so high that this approach will most likely be unsustainable in the long run. It is much more effective to dedicate time learning the language, so that in practice communicating will be much simpler, easier, and most of all: quicker.

In a similar vein, conventional flash algorithms, in and of themselves, are not that slow, and if you only need to perform them a (relatively) limited number of times, such as in most simple steady-state process simulations, the time they take up is acceptable. For these types of simulations conventional methods will suffice, just as looking up a few words here and there in the dictionary is fair when you're abroad only for a short period. However, just as moving to a foreign country will require you to look up words in the dictionary quite often, dynamic simulations require you to do many, many consecutive steady-state calculations, each containing again many flash calculations. And as a result, the time it takes to execute all of them will really start to add up. This problem too can be solved in a sense by "learning the language". Of course, there's no such thing as a flash calculation language, but the underlying concept of *front-loading* the work is the same: dedicating time and effort to the problem beforehand, so that in practice it takes much less of both. In the language case you invest time in remembering vocabulary and grammatical rules, while in the flash case you invest that time into gathering data, designing an appropriate neural network, and finally training the network. Once the time-consuming sourcing and learning process is over, using the network in a practical environment to make predictions can be done very quickly. This concept is illustrated schematically in Figure 1.2.

It should also be mentioned that just like a language, a properly trained neural network is very specific to a certain situation or problem. Even though you've spent months learning Spanish, it's not going to help you communicate with the locals in Poland. Similarly, a neural network trained to predict temperature values will be useless when used to predict values for pressure. On the other hand, the methods of designing and training a network *can* usually be used for different applications, just like the learning methods you have developed in order to teach yourself Spanish could be applied in a similar way to learn Polish.

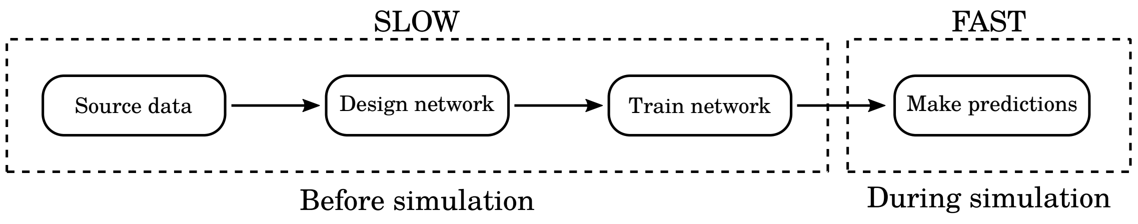


Figure 1.2: Schematic representation of the concept of front-loading work through the use of a neural network.

Chapter 2

Thermodynamics

In this chapter, background knowledge on the thermodynamics of vapor-liquid equilibrium will be provided. In addition, current methods of solving flash problems are described, and some of their shortcomings are discussed.

2.1 Basic concepts of thermodynamics

2.1.1 System

The word *system* is generally used to describe a separate part of the world which is the subject of the study at hand. A system can be anything physical or virtual, real or hypothetical, and can range from the size of a box containing a few molecules, to entire planets.

Anything not part of the system is referred to as its surroundings, and is separated from the system by a wall or boundary [13]. In thermodynamics, the system wall can either allow heat to be exchanged by the system and its surroundings, or not, in the latter case the system is said to be adiabatic [13]. If the walls also prohibit mass flow, the system is a closed system [13].

Throughout this text, the word system will most often be used to refer to (a mixture of) chemical compounds contained in a completely isolated box of a certain volume which may or may not be known. Schematic representations of such a system are shown in Figure 2.1.

2.1.2 State

A systems *thermodynamic state* is defined by the values of the systems properties, including among others pressure, volume, and temperature [14]. A systems state is uniquely defined by its property values, and if any of them change, the state of the system changes. A mathematical expression which defines the value of a state property is called a *state function* [14]. An expression which can be used to relate properties to each other is called an *equation of state* (EOS, see Section 2.1.8) [13–15].

A systems thermodynamic state also determines its *state of aggregation* (phase), i.e. whether it is solid, liquid, vapor, or in a state of equilibrium between any of these three [13, 14]. Figure 2.1 shows schematic representations of three phase types: pure vapor, pure liquid, and a two-phase vapor-liquid equilibrium. Property values depend quite a lot on the state of aggregation, so knowledge on a systems phase is very important.

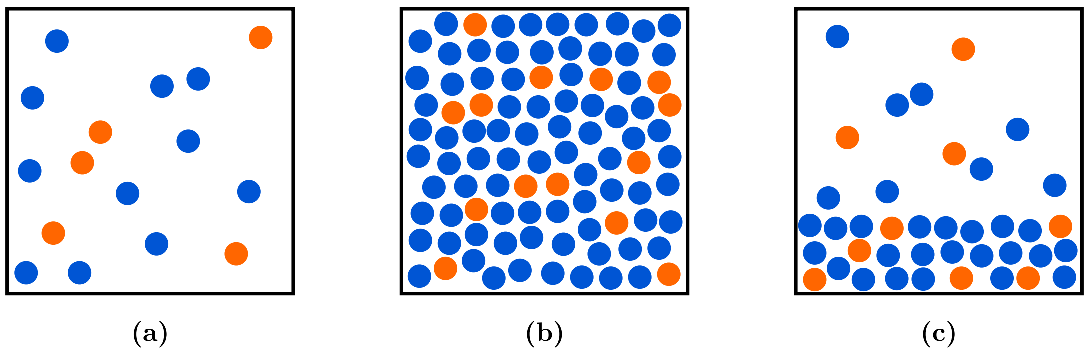


Figure 2.1: Schematic representation of three different states of aggregation (phases) of a binary mixture. The phases shown are: pure vapor (a), pure liquid (b), and a two-phase vapor-liquid equilibrium (c).

2.1.3 Thermodynamic properties

Thermodynamic properties are characteristics of a given system of chemical compounds which describe its *state*. Most thermodynamic properties are either directly measurable, or derivable from other properties. In this text, two groups of properties are of main interest, what are here referred to as *thermodynamic variables* and *energy functions*, which are briefly described in the next two sections.

Thermodynamic variables

In this text, the term *thermodynamic variable* is used to refer to thermodynamic properties whose combination can be used to derive almost all other thermodynamic properties [16]. The thermodynamic variables are: pressure (P), volume (V), temperature (T), entropy (S), chemical potential (μ), and (component) mole number (n), which expresses the amount of substance(s) present in the system.

Energy functions

Thermodynamic energy functions are used to express changes in energy of a system as a function of changes in its thermodynamic variables [14]. Four different energy functions exist: internal energy (U), enthalpy (H), Helmholtz free energy (A), and Gibbs free energy (G). Of the four energy functions, internal energy is the most fundamental, as the other three can all be derived from it [15]. The energy functions are expressed in the thermodynamic variables as follows:

$$U = TS - PV + \sum \mu_i n_i \tag{2.1a}$$

$$H = U + PV = TS + \sum \mu_i n_i \tag{2.1b}$$

$$A = U - TS = -PV + \sum \mu_i n_i \tag{2.1c}$$

$$G = U + PV - TS = \sum \mu_i n_i \tag{2.1d}$$

And their differentials:

$$\mathrm{d}U = T\mathrm{d}S - P\mathrm{d}V + \sum \mu_i \mathrm{d}n_i \tag{2.2a}$$

$$\mathrm{d}H = T\mathrm{d}S + V\mathrm{d}P + \sum \mu_i \mathrm{d}n_i \tag{2.2b}$$

$$\mathrm{d}A = -S\mathrm{d}T - P\mathrm{d}V + \sum \mu_i \mathrm{d}n_i \tag{2.2c}$$

$$\mathrm{d}G = -S\mathrm{d}T + V\mathrm{d}P + \sum \mu_i \mathrm{d}n_i \tag{2.2d}$$

2.1.4 Fundamental equations

Two related notions to the energy function are that of *natural variables* and *fundamental equations*. When an energy function is expressed as a function of three thermodynamic variables (one of which should always be the mole numbers), so that all other thermodynamic variables can be derived from the energy function through its partial derivatives, the expression for the energy function is referred to as a *fundamental equation*, the two variables (apart from mole numbers) which lead to a fundamental equation are referred to as its *natural variables* [15].

The natural variables of each energy function can be determined from the expression of its differential. For example, temperature and pressure can be determined by taking the partial derivatives of the internal energy with respect to entropy (at constant volume) and volume (at constant entropy) respectively. Thus, if the internal energy is known as a function of entropy and volume, temperature and pressure can readily be derived. The four fundamental equations with their natural variables and the expressions of the remaining thermodynamic variables are summarized in Table 2.1. With the results from Table 2.1 and Equations 2.1, the other three fundamental equations can be derived.

Table 2.1: Thermodynamic variables as derived from the fundamental equations [3, 15].

	P	V	T	S	μ
$U(S,V)$	$-\left(\frac{\partial U}{\partial V}\right)_{S,n_i}$	-	$\left(\frac{\partial U}{\partial S}\right)_{V,n_i}$	-	$\left(\frac{\partial U}{\partial n_i}\right)_{S,V,n_j}$
$H(S,P)$	-	$\left(\frac{\partial H}{\partial P}\right)_{S,n_i}$	$\left(\frac{\partial H}{\partial S}\right)_{P,n_i}$	-	$\left(\frac{\partial H}{\partial n_i}\right)_{S,P,n_j}$
$A(T,V)$	$-\left(\frac{\partial A}{\partial V}\right)_{T,n_i}$	-	-	$-\left(\frac{\partial A}{\partial T}\right)_{V,n_i}$	$\left(\frac{\partial A}{\partial n_i}\right)_{T,V,n_j}$
$G(T,P)$	-	$\left(\frac{\partial G}{\partial P}\right)_{T,n_i}$	-	$-\left(\frac{\partial G}{\partial T}\right)_{P,n_i}$	$\left(\frac{\partial G}{\partial n_i}\right)_{T,P,n_j}$

2.1.5 Intensive and extensive properties

Thermodynamic properties can be given either as *intensive* or *extensive*. The value of an extensive property depends on the size of the system, while the value of an intensive property does not [13, 14, 17]. For example, referring to Figure 2.2, given a system at a certain temperature and pressure at a constant volume, number of moles, and internal energy. If the system is divided into four (hypothetical) subsections, each subsection will have its own volume, number of moles and internal energy, which are all different from the original system. Therefore, volume, mole number and internal energy are all extensive properties. On the other hand, each subsection has the same temperature and pressure as the original system, which are therefore intensive variables.

Any extensive variable can be made intensive by dividing it by another extensive variable. For example, volume, expressed in units of m^3, and internal energy, expressed in units J, can be made intensive by dividing them by the number of moles, expressed in mol, changing their units to m^3/mol and J/mol respectively.

Throughout this text, most properties will be given in their intensive form (on a molar basis) unless otherwise specified. Normally, intensive variables are denoted with lower-case symbols, but in order to remain consistent with abbreviations for the different flash types, all properties in this text will be referred to by their upper-case symbols.

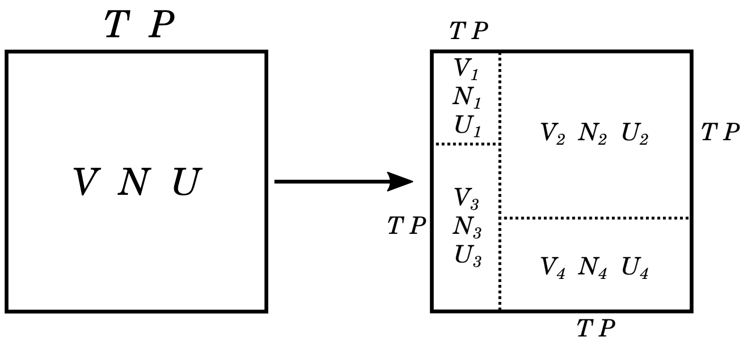


Figure 2.2: Schematic example illustrating the difference between intensive and extensive properties. Volume (V), number of moles (N), and internal energy (U) are extensive variables and thus are different for each hypothetical subsection in the system on the right. On the other hand, temperature and pressure are intensive variables, and therefore remain the same throughout the system.

2.1.6 Phase rule

As mentioned, the thermodynamic state of a system is defined by the values of all its thermodynamic properties. However, not all thermodynamic properties are independent of each other, and in most cases the values of only a few (intensive) properties need to be specified in order to fix the values of all remaining properties. The number of intensive properties that are required to do so is called the systems number of degrees of freedom, and can be determined using the Gibbs phase rule [13, 14]:

$$F = 2 - \pi + c \tag{2.3}$$

Here,

F	is the number of degrees of freedom,
π	is the number of phases in equilibrium, and
c	is the number of components present.

For example, the state of pure CO_2 ($c = 1$) in the vapor phase ($\pi = 1$) can be completely determined by specifying $2 - 1 + 1 = 2$ intensive properties, such as for example temperature and pressure. For multi-component mixtures, additional properties are required, which are most often given in the form of phase composition mole fractions [14].

In addition to Gibbs phase rule, Duhem's theorem states that when the individual amounts of all components present in a mixture are known, the (closed) system can be completely described by only 2 thermodynamic properties [14]. Furthermore, these properties are not restricted to being intensive, and can be extensive as well.

2.1.7 Equilibrium

A system is said to be in a state of equilibrium if no *macroscopic* changes occur within its boundaries. This means no flow of any physical quantity (heat, mass, charge) can exist in the system [13]. The word macroscopic gives an important distinction, as changes will still occur on a microscopic level: molecules still move around, collide, exchange energy, move from one phase to another, but when "zooming out", on average no changes occur as the behavior of one molecule is cancelled out by the opposite behavior of another.

In a thermodynamic system, equilibrium implies that the values of pressure (responsible for bulk flow), temperature (responsible for heat flow), and chemical potential (responsible for diffusion and reactions) are equal throughout the system [3, 13, 14]. The *fundamental postulate of thermodynamics* states that, given enough time, any isolated system will eventually reach a state of equilibrium [13].

Furthermore, depending on which two properties were used to fix the systems state, a certain state function will attain its minimum value at equilibrium. For example, a system fixed by its values of pressure and temperature will be at a minimum of its Gibbs free energy, while a system defined by its volume and temperature will attain the minimum value of its Helmholtz free energy function [3, 13].

2.1.8 The equation of state

An equation of state is an equation which relates the values of a systems thermodynamic properties to each other. The simplest equation of state is the ideal gas law, which is linear in both P, V, and T [13–15]:

$$P = \frac{RT}{V} \tag{2.4}$$

Here,

P	is the pressure in [Pa],
T	is the temperature in [K],
V	is the volume in [m^3/mol], and
R	is the universal gas constant (8.134 J/mol/K).

The ideal gas law assumes that the fluid molecules have no intrinsic volume and no interaction with other molecules. As a result, it is only valid at low to moderate pressures, and can only model the vapor phase, not the liquid phase. For this reason, the ideal gas law cannot be used to model vapor-liquid equilibrium, instead, cubic equations of states are used.

As the name implies, a cubic EOS has the form of a cubic polynomial, which makes it possible to model both the steep property changes of the liquid phase, as well as the gradual changes of the vapor phase. The first and most basic cubic EOS was developed by Van der Waals (VdW), who added two parameters to the ideal gas law, one (b) which models the fact that each molecule has its own volume, and another (a) which models the interactive forces between the molecules in the system [18]. The VdW EOS has the following shape:

$$P = \frac{RT}{V - b} - \frac{a}{V^2} \tag{2.5}$$

While the VdW EOS is a great improvement over the ideal gas law, it is not accurate enough for most practical problems, and other, more detailed cubic EOS are used. These include, among others, the Peng-Robinson [19], and Soave-Redlich-Kwong [20] equations of state. The Peng-Robinson EOS is one of the most widely used equations of state, and has the form:

$$P = \frac{RT}{V - b_{\text{mix}}} - \frac{a_{\text{mix}}}{V^2 + 2b_{\text{mix}}V - b_{\text{mix}}^2} \tag{2.6}$$

Here,

a_{mix}	is the mixture attraction parameter in [Jm^3/mol^2], and
b_{mix}	is the mixture intrinsic volume parameter in [m^3/mol].

The values of a_{mix} and b_{mix} depend on the composition and specific compounds present in the mixture, and their expressions are given in Appendix B. Figure 2.3 shows examples of the pressure as calculated from the Peng-Robinson EOS for pure water at three different temperatures. For the blue line, the leftmost part of the curve, where the line is steepest, corresponds to the liquid phase region, while the right part, where the slope is more gradual, corresponds to the vapor phase region. Determining the pressure for the two-phase vapor-liquid equilibrium region is not as simple as plugging the volume into an EOS, this is explained in the next section.

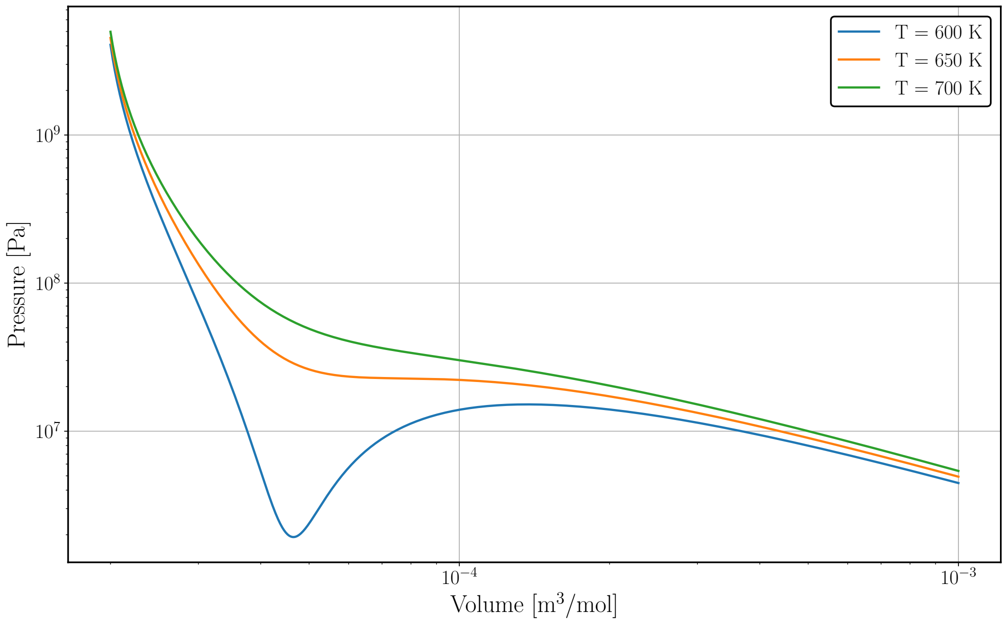


Figure 2.3: Plot of the pressure of pure water, calculated using the Peng-Robinson equation of state [19] for three different temperatures.

2.2 Vapor-liquid equilibrium

In the previous sections it was mentioned that a systems state also defines the systems phase, i.e. whether it is solid, liquid, or vapor. However, a system can also be in a state of equilibrium of two or more phases. If this is the case, the equilibrium conditions state that the pressure and temperature must be equal for all phases, the chemical potential of each component in each phase must be equal, and that the relative amounts of both phases should remain constant [3, 14]. However, extensive properties need not be equal for both phases. For example, each phase can have a different volume, number of moles, and internal energy.

Additionally, even the distribution of each separate component need not be equal, and it can occur that most of a certain components molecules will be in one phase while only very few will be in the other. In fact, most separation processes (such as distillation) rely heavily on the fact that some components have a clear “preference” for one phase over another [1].

On the other hand, all laws and rules described in the last section do still hold true: fundamental equations can still be used to derive all other properties, Duhem’s theorem remains valid, and the system will still attain a state function minimum at equilibrium.

2.2.1 Concepts and notation

The composition of a mixture of chemical compounds indicates how much of each component is present. Composition is most often expressed on a molar basis, and can be done either in an absolute manner, by specifying the mole number of each component, or a relative manner, by giving the mole fraction of each component. The sum of all mole fractions should always add up to one. This can be summarized in equation form as [3, 13–15]:

$$N^{\mathrm{F}} = \sum_{i=1}^{c} n_i^{\mathrm{F}} \tag{2.7a}$$

$$z_i = \frac{n_i^{\mathrm{F}}}{N^{\mathrm{F}}} \tag{2.7b}$$

$$\sum_{i=1}^{c} z_i = 1 \tag{2.7c}$$

Here,

N	is the total number of moles in [mol],
n_i	is the mole number of component i of the mixture in [mol],
z_i	is the mole fraction of component i of the mixture in [mol/mol], and
c	is the total number of components present in the mixture.

The superscript F in the above equations stands for *feed* and indicates that the property/variable in question pertains to the combined total of the entire system.

The same rules and notation as above apply to the individual phase compositions, with z replaced by y and x, and F by V and L for the vapor and liquid phases respectively. The ratio between the total number of moles in the vapor phase, and the total number of moles in the liquid phase is called a mixtures vapor fraction:

$$\beta = \frac{N^{\mathrm{V}}}{N^{\mathrm{F}}} \tag{2.8}$$

Here, β is the mixtures total vapor fraction in [mol/mol].

In a similar vein, the ratio between a components mole fraction in the vapor phase and its mole fraction in the liquid phase is referred to as its equilibrium constant:

$$K_i = \frac{y_i}{x_i} \tag{2.9}$$

Here,

K_i	is the equilibrium constant of component i in the mixture,
y_i	is the mole fraction of component i in the vapor phase in [mol/mol], and
x_i	is the mole fraction of component i in the liquid phase in [mol/mol].

When a mixtures vapor fraction and distribution coefficients are known, as well as its feed composition, the mole fractions in the liquid and vapor phase can be calculated through the so-called Rachford-Rice equations [21]:

$$x_i = \frac{z_i}{1 + \beta(K_i - 1)} \tag{2.10a}$$

$$y_i = x_i K_i \tag{2.10b}$$

Balance equations

A system at vapor-liquid equilibrium must satisfy a number of balance equations that insure conservation of mass and energy [3, 14]. The first equation is the mass balance equation:

$$N^{\mathrm{V}} + N^{\mathrm{L}} = N^{\mathrm{F}} \tag{2.11}$$

Which says that the total amount of moles in the vapor and liquid phases together should equal the total amount of moles in the feed. Another way of saying this is that the phase compositions y_i, x_i, and z_i should all add up to one (as per Equation 2.7c).

In addition to the mass balance equation, the component balance equation:

$$n_i^{\mathrm{V}} + n_i^{\mathrm{L}} = n_i^{\mathrm{F}} \tag{2.12}$$

Specifies that the total amount of a component in the vapor and liquid phase together should equal the amount of that component in the feed.

Lastly, the energy balance equation insures that the enthalpy in both phases together is the same as the enthalpy of the feed (assuming no heat is added or removed from the system):

$$H^{\mathrm{V}} + H^{\mathrm{L}} = H^{\mathrm{F}} \tag{2.13}$$

Here, H is enthalpy in J/mol or J.

2.2.2 Flash calculations

According to Duhem's theorem explained in Section 2.1.6, when a systems composition is known, only two thermodynamic properties are required to fix the values of all other properties of the system. Any calculation method that takes in the values of two thermodynamic properties and composition of a system and determines the values of the remaining properties is called a *flash* calculation (or simply *flash*). Usual notation puts the symbols representing each of the two properties in front of the word *flash*, for instance, a flash calculation which starts with values for pressure and temperature is called a *PT*-flash. In the case of a mixture of compounds, the composition of the mixture must always be known, for this reason it is not always included in the name of a flash, as it is implied.

In theory any two properties can be specified, but in practice, certain combinations result in much easier calculations, while others have no to very limited applications. Examples of flash calculations include:

- The pressure, temperature, PT-flash [22].
- The vapor fraction, temperature/pressure $\beta T/P$-flash [7].
- The pressure, enthalpy, PH-flash [23].
- The pressure, entropy, PS-flash [23].
- The volume, temperature, VT-flash [24].
- The internal energy, volume, UV-flash [25].
- The entropy, (stagnation) enthalpy, SH-flash [8].

As the goal of a flash calculation is finding the equilibrium properties of a given system, many flash algorithms work by minimizing an appropriate state function [3,26]. The next section briefly covers a simple PT-flash algorithm, which will serve to illustrate the basic theory behind solving vapor-liquid equilibrium problems. Examples of some more specific and complex algorithms are given in [8,21,24–28].

The basic vapor-liquid flash

When a system is either in a state of pure liquid, or pure vapor, an EOS can be used to to calculate all properties of the system using the values of the feed composition (z_i) and the two specified properties. On the other hand, when a system is in vapor-liquid equilibrium, an EOS cannot be used directly as the compositions of each phase (x_i and y_i) are not known beforehand.

A vapor-liquid flash algorithm thus iterates on the values of the vapor and liquid compositions until both the equilibrium conditions (equal pressure, temperature and chemical potential in both phases) and the balance equations (mass and energy of both phases together should equal the feed mass and energy) are satisfied. An example PT-flash algorithm is used here to illustrate these basic principles, although no energy balance is explicitly solved as enthalpy was not one of the two input variables. For an example PH-flash, see [23].

The algorithm works by splitting the feed into two phases (vapor and liquid), and iterating on the vapor fraction and equilibrium ratios $K_i = \frac{y_i}{x_i}$ values until the fugacities of all components in both phases are equal, making sure mass and component balances are not violated.

Fugacity is an effective partial pressure which is often used to replace chemical potential in equilibrium calculations as it has certain characteristics that make it more easy to work with [14]. In multi-phase equilibrium, the fugacity of each component should be equal in every phase.

The PT-flash algorithm takes the following steps [3]:

1. An initial guess on the K_i values is determined based on the specified input pressure and temperature, and Wilson's correlation [29]:
$$K_i = \frac{P_{\mathrm{c},i}}{P} \exp\left[5.42\left(1 - \frac{T_{\mathrm{c},i}}{T}\right)\right] \tag{2.14}$$
Here, $P_{\mathrm{c},i}$ is the critical pressure of component i, and $T_{\mathrm{c},i}$ is the critical temperature of component i. Once the initial guess has been generated, an iteration loop is started in which the following steps are taken:

2. Based on the current values of K_i and the original feed mole fractions z_i, the vapor fraction β is calculated by solving the equation:
$$\sum_{i=1}^{c} \frac{z_i(1-K_i)}{1+\beta(K_i-1)} = 0 \tag{2.15}$$

3. Using the current K_i, vapor fraction β and original feed mole fractions z_i, the mole fractions in both phases are calculated using the equations:
$$x_i = \frac{z_i}{1+\beta(K_i-1)} \tag{2.16a}$$
$$y_i = x_i K_i \tag{2.16b}$$

4. Using an equation of state, the fugacities of all components in both phases are calculated based on the mole fractions calculated in the previous step, see Appendix B for the exact mathematical expressions.

5. The K_i values are updated as follows:
$$K_i^{k+1} = K_i^k \frac{f_i^{\mathrm{L}(k)}}{f_i^{\mathrm{V}(k)}} \tag{2.17}$$
Here, k is the iteration number, and f_i^{L} and f_i^{V} are the fugacities of component i in the liquid and vapor phase respectively.

6. Convergence is checked by comparing the fugacities in each phase, if they are equal to within a predefined precision, iterations are stopped and the current values of β, x_i, and y_i are set as the final values. If convergence is not reached, steps 2-5 are repeated until it is reached.

Once the vapor and liquid phase compositions are known, an equation of state can be used to calculate the value of most other properties such as volume, entropy, enthalpy, internal energy, Gibbs free energy, and Helmholtz free energy [3].

The above steps are similar for most vapor-liquid equilibrium algorithms, but more (complex) steps can be added to increase robustness or convergence speed. For example, the value of a corresponding state function that should attain its minimum at equilibrium is often calculated to insure the algorithm is going towards to correct solution at every iteration [22, 24, 25]. Furthermore, the first and second derivatives of said state function can be calculated to determine the precise compositional changes required to insure an efficient direction towards the minimum [24, 25]. A large number of other solution approaches exist, of which a selection can be found in [27, 28, 30–32].

The precise algorithm used by the TEA property calculator is described briefly in Appendix A.

2.2.3 Phase stability analysis

Equilibrium calculation are only required when a given system is in vapor-liquid equilibrium. If the system is in a state of pure vapor or pure liquid, equilibrium calculations will be unnecessary, and property values can be calculated relatively straight-forward from an equation of state without the need for iteration on phase compositions [3]. In fact, often times an equilibrium calculation algorithm that is given inputs which actually specify a single phase state, the algorithm will return incorrect results [30].

Therefore, in order to prevent unnecessarily executing a vapor-liquid equilibrium calculation algorithm, most flash algorithms will include a so-called *phase stability test*, which determines whether a given mixture is in a state of multi-phase equilibrium, or in a pure single phase state. In fact, the majority of the total calculation time in a flash algorithm is spent on stability analysis [33].

Depending on the flash type, different types of stability criteria and algorithms are used. For example, most PT-flash types will apply the widely used tangent plane criterion approach developed by Michelsen [29], while a VT-flash might use an adapted version of the tangent plane criterion, or a specialized VT-flash stability algorithm [34, 35]. Other stability criteria can be found in [3, 36–39].

Most of the algorithms referenced above are based on a partial state function minimization approach. As was previously mentioned, a system is at its equilibrium state when the value of a certain (fundamental) state function is at a minimum. For example, a system at specified pressure and temperature will be at equilibrium when the value of its Gibbs free energy is at a minimum [3].

Many stability tests check to see whether a multi-phase composition exists at the given values of the input variables for which the value of the fundamental state function is lower than the value of that state function at a state of pure phase. If such a multi-phase composition exists, it must therefore mean that the single phase alternative cannot be the systems equilibrium state as it is not at a position where its fundamental state function attains a minimum (since a different multi-phase position exists with a lower value).

In theory, the method used above requires checking every possible potential multi-phase equilibrium state for the value of its fundamental state function, as there might always be a certain potential multi-phase state that wasn't checked but might have a value of the fundamental state function lower than the single phase case. In practice this would require global optimization of the state function in question, which is computationally very resource intensive [40].

Luckily, most times only a few well-chosen multi-phase trial-compositions need to be checked in order to give a satisfactory result in most cases. For instance, this limited trial-phase approach is used in [29, 34], considerably reducing their computational load.

2.3 Flash algorithm complications

In this section a brief overview is given of complications that can arise in conventional flash algorithms, mainly focussed on problems regarding robustness (failure to find correct solutions) and computational speed.

2.3.1 Robustness complications of flash calculations

There are a few main situations in which a flash calculations may fail, these are:

- Phase stability analysis might fail near points on the so called stability test limit locus (STLL), where convergence slows down considerably, or diverges altogether [41]. In addition, the existence of a “shadow-region” can further complicate the stability analysis [33].
- Both the flash equilibrium algorithm and the phase stability algorithm require an initial guess [3], a starting point from which the algorithm begins. If the initial guess is not precise enough, the algorithm can converge to a local minimum of the state function instead of the global minimum, thus finding an incorrect equilibrium position [36, 42]. For more details on optimization and the difference between local and global minima see Section 3.4.2 and Appendix D.
- If the values of the input properties given are attained at or near the critical point of the system, calculation methods may either take much longer to converge, or fail to converge at all. Moreover, if converged, the likelihood that the solution found is a trivial solution is much higher [7].
- If input properties given are very close to the boundary between two phase regimes (for example the transition from pure vapor to vapor-liquid equilibrium) many algorithms show similar problems as near the critical point [8].
- Again near phase boundaries, a situation may occur in which a stability analysis incorrectly indicates whether a given system in in a state of vapor-liquid equilibrium or a single phase state. In this case, following fluid property value predictions will be incorrect. While this does not directly lead the algorithm to crash, if the mispredicted values are used in a process simulator, it might lead to divergence in later stages of the simulation.

The final two situations are especially troublesome in dynamic simulations of system in which phase transitions occur, and flash calculation near phase boundaries are frequent and often unavoidable.

2.3.2 Speed complications of flash calculations

Predicting vapor-liquid equilibrium requires iteration on the vapor and liquid phase compositions until a solution is reached which satisfies all equilibrium conditions. Naturally, iterating until convergence will take longer to compute than an explicit input-output relation, as most calculations will have to be repeated a (large) number of times. In some cases, phase stability and flash calculations can take up to 50-70% of the entire simulation time [4, 5]. In addition, many conventional solution algorithms, such as the one used by TEA [11, 31] do not allow for the processing of multiple input combination at the same time, meaning that when a large number of flashes has to be executed, each flash has to be solved separately.

Actual calculation times will be more thoroughly detailed in Section 5.1, but back-of-the-envelope estimate is given here to indicate the effect calculation speed can have on the total duration of a dynamic process simulation.

Assuming that a single flash calculation takes an average of 0.001 seconds, the entire system of interest is discretized into 1000 fluid packets for which fluid properties will have to be calculated, a single day of 7 hours of operation is modelled, and a (generous) time step of 0.1 seconds can be used to properly model the (stiff) dynamic behavior of the system, it will take approximately: $0.001 \times 1000 \times \frac{7}{0.1} = 70$ hours to model just the fluid properties within the system over a single day of operation. Even if a speed increase of around 10 times can be achieved, calculation times will now be on par with the actual operation time of the system. A serious improvement.

2.3.3 Benefits of artificial neural networks

The previous sections mentioned a number of complications that can occur in conventional flash solution methods. Here, potential benefits that can be obtained from the application of artificial neural networks are listed.

- Neural networks, once trained, require no iterative process to arrive at a solution [9, 43, 44]. Therefore, any complications surrounding the slowing down or failing of convergence can be avoided.

- In addition to the previous point, the fact that neural networks do not require any iterative processes also means that they are independent of an initial guess, averting any complications that can arrive from incorrect initial guess estimations.

- Neural networks, once trained, only require matrix multiplication and addition, and the execution of basic mathematical expressions to map an input to an output [9, 43, 44]. These mathematical operations are fast and can easily be parallelized [9, 43], meaning neural networks can process multiple input combinations at once.

- Neural networks do not require an understanding of underlying physical principles in order to be able to be used to correlate different variables to each other [9, 43]. As a result, the same framework can be used to solve multiple different types of flash problems, without the need for rewriting (parts of) the governing equations and solution algorithms.

Chapter 3

Artificial Neural Networks

Artificial Neural Networks (ANNs) are a broad array of *machine learning* calculation methods inspired by the way in which animal brains function and learn. While the original concepts and theory of the artificial neuron and the neural network were developed during the 1940s-1970s, and later refined and improved upon during the 1980s and 1990s [45], the advent and increased availability of powerful computational hardware has led to large increase in the application of ANN in the past 10-20 years. Nowadays, ANN are utilized for many different applications in a wide variety of fields [46–52].

This chapter aims to give an introduction into the theory and mathematics behind ANN, explain in more detail one specific type of AN, and finally discuss the main methods of *training* ANN. The chapter finishes with a short overview of existing applications of ANN to property predictions and vapor-liquid equilibrium problems.

3.1 Neural network basics

The most basic ANN is made up of a network of units called nodes or neurons and the connections between them. The neurons are divided into multiple separate layers, each neuron in one layer is connected to every neuron in the layer directly before and after it. Neurons in the first layer of a network are only connected to neurons in the next layer, neurons in the last layer of the network are only connected to neurons in the previous layer. The first and last layers in a network are usually referred to as the *input layer* and *output layer* respectively. The remaining layers in a network are referred to as *hidden layers* [9, 43, 53].

Every neuron in a neural network has a corresponding internal value called its *activation* A_i. This activation is propagated as information signals from one neuron to another. In many networks, information can only be passed on in a forward direction, from one layer to the next layer in the network, but some more complicated architectures also allow for information to be passed backwards, from one layer to a layer that comes before it. The first type of network is called a feedforward neural network (FNN), the latter type a recurrent neural network (RNN) [44].

The amount of layers in a network, how many neurons there are in each layer and how they are connected to other neurons in the network is often called the *topology* of the network. An example topology of a simple feedforward neural network is given in Figure 3.1.

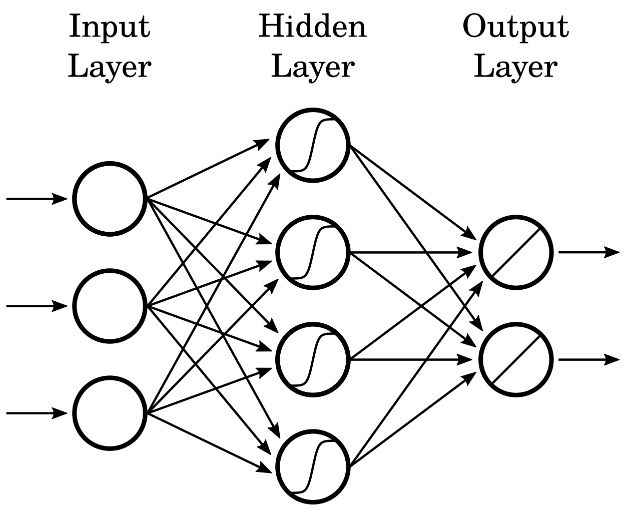


Figure 3.1: Schematic representation of a simple example neural network consisting of three input neurons, four hidden neurons, and two output neurons. Every neuron in one layer is connected to all neurons in the next layer. Information is propagated only in the forwards direction, from the input layer, through the hidden layer, to the output layer.

In a typical ANN, information is fed into the network through the input neurons, which pass on the information to the neurons in the hidden layer(s). The information is processed in the hidden neurons and again passed on to either the next hidden layer, or if the current hidden layer is the last one, to the output layer. In the output neurons, the information is processed one final time, and the processed information is considered the output of the ANN.

Every ANN has certain aspects that determine the exact relation between input and output of the network (based on a certain input, what will the output of the network be). These aspects are [9, 43, 44, 53]:

- Weights and biases.
- Activation rules and functions.
- Topology.

In the remainder of this section, the aspects mentioned above will be discussed.

3.1.1 Weights and biases

In an ANN, every connection between a neuron i and a neuron j has a corresponding weight w_{ij} [9, 43, 44, 54]. The weight of a neuron pair connection determines how strongly the information of one neuron in the pair is passed on to the other neuron in the pair. Figure 3.2 gives a graphical interpretation of the weights between multiple neurons.

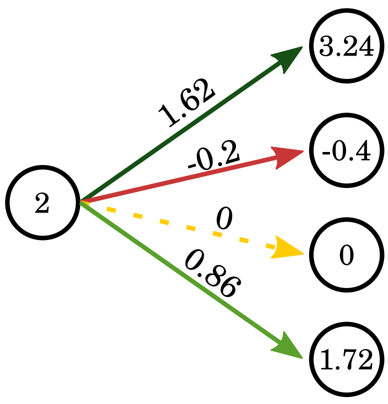


Figure 3.2: Graphical representation of the weights of the connections between pairs of neurons. Green arrows indicate connection with positive weight, while the red arrow indicates a connection with a negative weight. The yellow arrow means that the weight of that connection is zero, and thus no information is propagated through it.

The value of each neuron in the second layer is directly dependent on the value of the neuron in the first layer and the weight of the connection between the neurons. Furthermore, as can be seen from Figure 3.2, a weight can have any real value, be it positive, negative or zero. A connection with a negative weight means that the the first neuron has an inhibiting effect on the second neuron in the pair, not that the direction of propagation is reversed (weight is a scalar, not a vector). A weight of zero is the mathematical way of indicating two neurons are not connected but the potential for a connection still exists.

The matrix $\boldsymbol{W}$ is the matrix containing all weights connecting the neurons in one layer to the neurons in the next layer. For a simple feedforward network as shown in Figure 3.3a $\boldsymbol{W}$ typically has size $\mathbb{R}^{n \times m}$, where n is the number of neurons in the first layer, and m is the number of neurons in the second layer, while for a recurrent network such as shown in Figure 3.3b where neurons within a layer can be connected to each other, themselves or back to the previous layer, $\boldsymbol{W}$ can have a size of up to $\mathbb{R}^{(n+m) \times (n+m)}$, or even higher when connections to layers further back than one are taken into account.

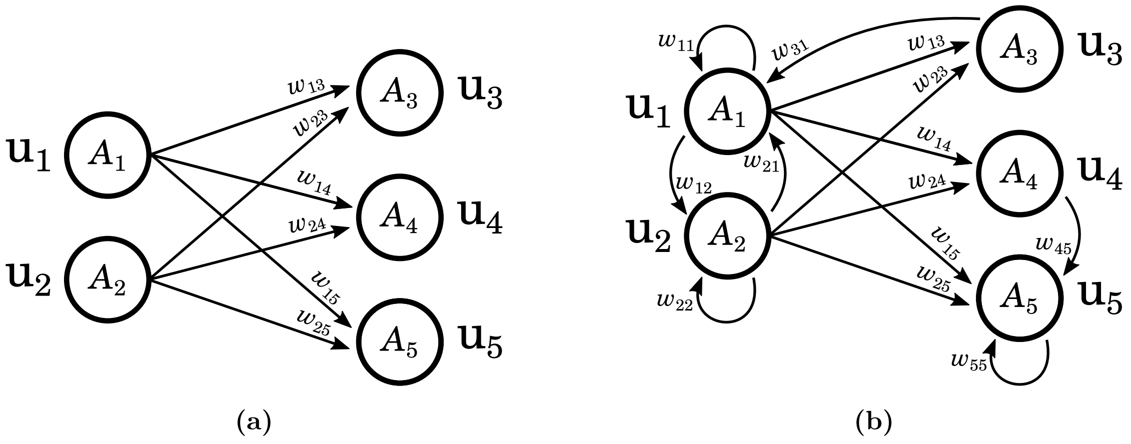


Figure 3.3: Example of two connected feedforward layers (a), and two connected recurrent layers (b).

Here is probably a good place to discuss some nomenclature with regards to neural network layers used in this text. When two connected layers in a feedforward network (such as in Figure 3.3a) are discussed, the first layer (depicted as the left layer) will be referred to as the *propagating layer*, as in this context it is the layer propagating information forward. In contrast, the second layer (depicted on the right) is referred to as the *receiving layer*, as it receives information from the previous layer. It should be noted that in the context of an entire neural network, every hidden layer is both a receiving and a propagating layer (it will first receive information from the previous layer, before propagating it forward to the next layer), whereas the input layer can only be a propagating layer and the output layer can only be a receiving layer. Therefore the terms receiving layer and propagating layer should only be used when discussing the connections and relations between two layers, and not the neural network as a whole.

Unless otherwise specified, when weight matrices are discussed in this text, it will be in the context of feedforward networks of size $\mathbb{R}^{n\times m}$. As an example, the weight matrix corresponding to the network in Figure 3.3a has the following form:

$$\boldsymbol{W} = \begin{bmatrix} w_{13} & w_{23} \\ w_{14} & w_{24} \\ w_{15} & w_{25} \end{bmatrix}$$

In general, the $(ij)^{\text{th}}$ element of a weight matrix $\boldsymbol{W}$ is the weight w_{ij} connecting the i^{th} neuron in the propagating layer to the j^{th} neuron in the receiving layer.

In most neural networks, every neuron has a corresponding *bias* term (b_i) [9, 43, 44, 53]. The bias can be considered as a constant input that is unique for every neuron in the system, a neurons bias does not depend on any other part of the network [9, 44]. In other words, a bias can be viewed in a way as a neuron with a certain value and a fixed weight of one, which is only connected to a single other neuron in the network, this is shown in Figure 3.4. Inversely, the bias term can also be interpreted as a neuron with constant fixed value of one, and certain weight value. However, in most graphical representations the bias terms are left out to prevent over-complicating the image.

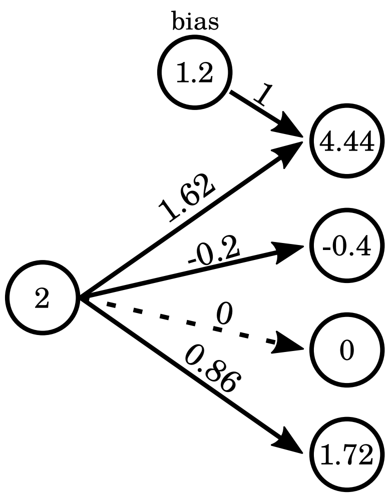


Figure 3.4: Graphical interpretation of weights of connections between pairs of neurons including the bias term of one of the neurons. As shown, the bias term adds a constant value to the total signal received by the neuron.

The vector $\boldsymbol{B}$ is a (column) vector containing the biases of all neurons in a single layer and thus has a size $\mathbb{R}^{n\times 1}$, where n is the number of neurons in the layer. Again, when biases are mentioned in this text it will be in the context of a feedforward network, unless otherwise specified.

As an example, the bias vector of the receiving layer of the network shown in Figure 3.3a has the following shape:

$$\boldsymbol{B}_{\mathrm{r}} = \begin{bmatrix} b_3 \\ b_4 \\ b_5 \end{bmatrix}$$

In general the i^{th} element of a bias vector $\boldsymbol{B}$ is the bias b_i corresponding to the i^{th} neuron in the layer.

As opposed to regular fitting methods where the fitting parameters often have some sort of physical meaning, the weights and biases in a neural network are purely mathematical constructs (however, inspired by a physical system). As a result, they cannot be given any meaningful initial value, and are usually randomly initialized within a certain interval [9, 55].

3.1.2 Activation

As mentioned in the previous section, every neuron in a network has a corresponding value called its *activation* A_i. The activation of a neuron can be seen as its activity or strength within a network, and it is the “information” which is propagated to the next layer in the network. The higher a neurons activity, the bigger its influence on the neurons it is connected to (depending on its weights). As can be seen from Figure 3.4, a neurons activation is directly related to the activation of the neurons it is connected to in the layer before it, the weights corresponding to those connections, the neurons bias and its *activation rule* and *activation function* [9, 44].

A neuron will typically receive “signals” (activation times weight) of many other neurons. All these signals from separate neurons will form one single total signal in the receiving neuron, how this signal is formed depends on the neurons activation rule. One of the most used activation rules is that of simple, straight-forward summation: all the received signals are summed together to form one total signal:

$$S_j = b_j + \sum_{i=1}^{n} s_{ij} = b_j + \sum_{i=1}^{n} A_i w_{ij} \tag{3.1}$$

Or in matrix notation:

$$\boldsymbol{S}_{\mathrm{r}} = \boldsymbol{B}_{\mathrm{r}} + \boldsymbol{W}\boldsymbol{A}_{\mathrm{p}} \tag{3.2}$$

Here,

S_j	is the total signal received by neuron j in the receiving layer,
b_j	is the bias corresponding to neuron j,
s_{ij}	is the signal received by neuron j in the receiving layer from neuron i in the propagating layer, and
n	is the total number of neurons in the propagating layer.

$\boldsymbol{S}$, $\boldsymbol{B}$, $\boldsymbol{W}$ and $\boldsymbol{A}$ are the signal vector, bias vector, weight matrix, and activation vector respectively. The subscript r stands for the receiving layer and p stands for the propagating layer.

Other activation rules used include the multiplication rule, where all signals are multiplied together to form the total signal, and the max-rule, in which the total signal is set as the strongest signal received from the propagating layer [44]. However, these are rarely used in practice.

It is important to note that there is a difference between the total received signal of a neuron and the activation of a neuron. The activation is the value of the neuron after an *activation function* $f_\mathrm{a}()$ is applied to its total received signal:

$$A_i = f_\mathrm{a}(S_i) \tag{3.3}$$

The activation function used should be non-linear in order to properly model the behavior of biological neurons and to be able to solve non-linear problems [56]. The linear mapping in which the activation is equal to the received signal ($A_i = f_\mathrm{a}(S_i) = S_i$) is called the identity mapping/function, or the linear activation function.

Examples of widely used non-linear activation functions are the (logistics) sigmoid function, the tanh function, the Rectified Linear Unit (ReLU), and the leaky ReLU [9], which are all shown in Figure 3.5.

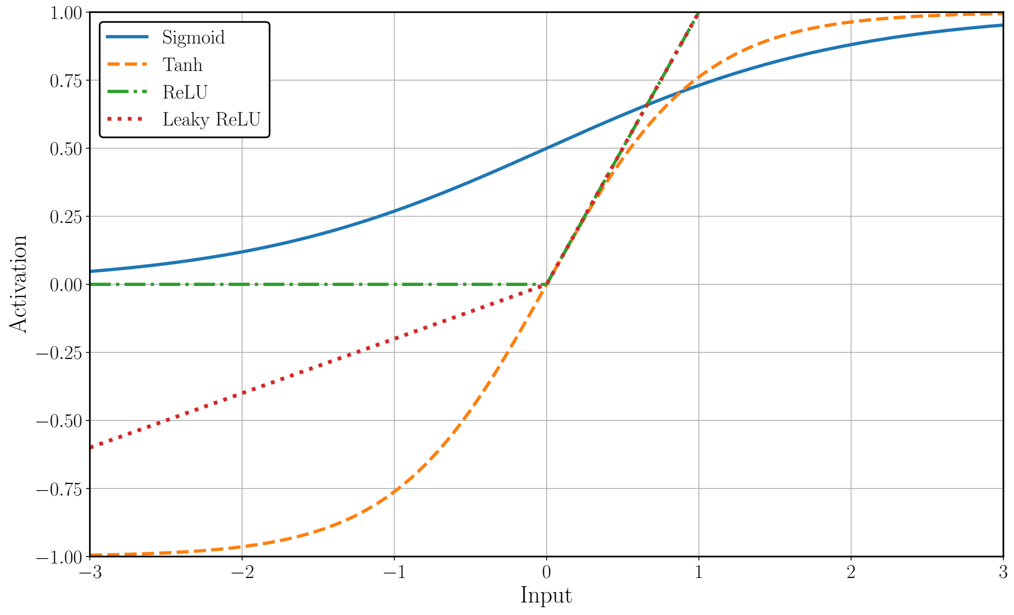


Figure 3.5: Examples of commonly used activation functions. Shown here are the sigmoid function (solid blue line), the tanh function (dashed orange line), the Rectified Linear Unit function (dash-dotted green line), and the leaky Rectified Linear Unit (dotted red line) [9].

The entire process of signal propagation from neurons in one layer to one of the neurons in the next layer is expressed mathematically as:

$$A_j = f_\mathrm{a}(b_j + \sum_{i=1}^{n} A_i w_{ij}) \tag{3.4}$$

Or again in matrix notation:

$$\boldsymbol{A}_r = f_\mathrm{a}(\boldsymbol{B}_\mathrm{r} + \boldsymbol{W}\boldsymbol{A}_\mathrm{p}) \tag{3.5}$$

This is graphically represented (without the bias terms) in Figure 3.6.

As an example, the activation of the second layer in Figure 3.3a is:

$$\begin{bmatrix} A_3 \\ A_4 \\ A_5 \end{bmatrix} = f_\mathrm{a}\left(\begin{bmatrix} b_3 \\ b_4 \\ b_5 \end{bmatrix} + \begin{bmatrix} w_{13} & w_{23} \\ w_{14} & w_{24} \\ w_{15} & w_{25} \end{bmatrix}\begin{bmatrix} A_1 \\ A_2 \end{bmatrix}\right) \tag{3.6}$$

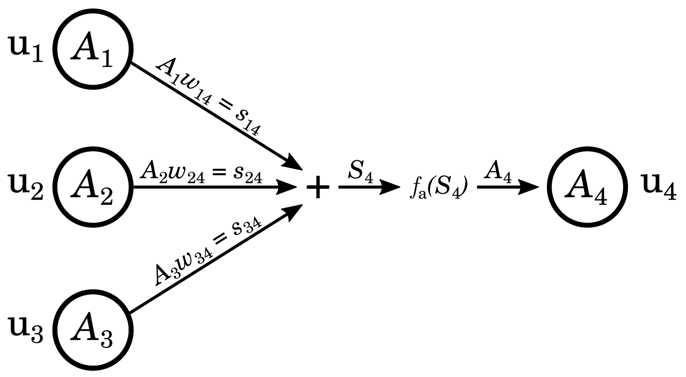


Figure 3.6: Graphical representation of signal propagation between neurons in one layer to a neuron in a subsequent layer.

3.1.3 Topology

The number of layers, neurons in a layer, and the connections between the neurons define the topology of a network. In general, a neural network can solve any non-linear problem [57], and different topologies can be used to solve the same problem. However, accuracy and computational speed can vary significantly from one topology to another. A network with too few neurons might have trouble solving a given problem, while a network with too many neurons may overfit to the training data, and fail in generalizing to new data points [45]. A *good* network topology has high accuracy with little complexity.

Finding the *best* topology for a given problem is a complex problem and is either done manually through trial-and-error or through a Genetic Algorithm (GA). The trial-and-error method is often done for simple problems in which finding the optimal topology is not of crucial importance, GAs are applied when the problem (and network) are very complex and finding a very good topology is important to decrease computational intensity.

3.2 The Multi-Layer Perceptron

Probably the most basic artificial neural network architecture is that of the Multi-Layer Perceptron (MLP). The MLP is a simple feedforward neural network with at least three layers (input, one hidden, and output) of any number of neurons containing any type of activation function [9]. The network shown in Figure 3.1 is an example of an MLP.

3.2.1 Universal Approximation Theorem

According to the Kolmogorov-Arnold theorem (of which a proof is given in [58]), any *multivariate*, *continuous* function can be approximated by a superposition of continuous functions of two variables. Based on this theorem, George Cybenko proved in 1989 that any MLP with a single hidden layer using sigmoidal activation functions can be used to approximate any continuous function to any arbitrary accuracy [59]. This is called the *universal approximation theorem* (UAT) of artificial neural networks. Kurt Hornik later proved the UAT for activation function other than sigmoidal ones [60]. More specifically, he proved that the UAT holds for any *continuous*, *bounded*, and *non-constant* activation function [60], such as the sigmoid, but also the tanh function. More recently the UAT was proved for any *unbounded* activation function [61], such as the Rectified Linear Unit (ReLU). Because of the UAT, neural networks are often called *universal approximators.*

Both Cybenko and Hornik mention that while the UAT states that for every continuous problem there exists a network that can solve it to any desired accuracy, it does not specify how to find that network [59, 60]. It is generally believed that finding the right network (correct topology, weights, and biases, etc.) that solves a given problem is an NP-complete problem [53, 54], meaning that it is easy to verify whether a solution is correct or not, but not easy (extremely hard) to find a solution in the first place (for instance, the solution to a sudoku puzzle can be quickly checked whether to be correct, but actually finding the solution takes a relatively long time, making it an NP-complete problem [62]). The UAT is one of the main reasons that neural network have found such widespread application.

3.2.2 Multi-Layer Perceptron applications

In this section a brief overview is given of general applications of the Multi-Layer Perceptron neural network

Supervised learning applications

In supervised learning, a neural network is given a large collection of inputs with known corresponding outputs, called *targets* [9]. In this kind of learning the training data set consists of input values with an output value corresponding to those input values, through the learning process the network will *learn* to match inputs with their appropriate outputs [57]. The main supervised learning applications are classification and regression.

Given a set of (multivariate) data instances in which each instance belongs to one of a number of discrete classes. The act of distinguishing between these output classes based on the values of the input variables is known as classification [57].

A well known example of classification within the world of machine learning is that of the iris flower data set [63]. This data set contains 150 measurements of the lengths and widths of the sepal and petal parts of three species of iris flowers. Disregarding outliers, a flower can be neatly classified as one species based on the value of its sepal and petal lengths and width. Thus, the classification problem becomes determining which species an iris belongs to, solely based on the lengths and width of its sepal and petal parts.

Image recognition is another example of a classification problem, be it more complicated as it deals with more complicate input data (images) and often has many possible outcome classes.

Where classification deals with predicting one instance out of a discrete set of possible outcomes, regression aims to predict the value of a continuous variable based on the given value(s) of one or more other continuous variables [9].

An example of regression is predicting the potential second-hand resale value of a specific model car based on its current mileage, age, and original selling price.

Unsupervised learning applications

In unsupervised learning, a network is trained on a given set of inputs which do not have a known corresponding output. The goal of the learning process is not to predict a known value or class, but to discover previously unknown structures/correlations in the data set [9, 57]. The learning process is used to unearth relationships in the data set that were not known beforehand, and the found relationships must often be interpreted separately in order to connect some meaning to them. Unsupervised learning applications include clustering and dimensional reduction.

Clustering is used to find previously unknown groupings or classes in large data sets [57]. For example, a web-shop might use an unsupervised learning neural network to find groups of users who typically buy the same things in order to more effectively recommend new products or services to their users.

When given a very large data sets containing many different features, a neural network can be used to reduce the data sets dimensionality to include only the most important features found in the data or generate a smaller-sized representation of the data set (comparable to zipping a data file) [57].

3.3 Data processing

Example data form the foundation of any sort of machine learning, including artificial neural networks. Proper data management can make the difference between decent or great neural network performance. In this section, the different steps required to go from raw data to training-ready data are described.

3.3.1 Training, testing, and validation data sets

In order to properly train, evaluate, and choose between different neural network alternatives, it is important to divide the total data set available into three subsets: the training set, the testing set, and the validation set [9].

Training set

The training set is the part of the original data set which is used to train the neural network. One of the most important aspects of the training set is that it should be representative of the entire data set. Without a representative training set, the network will develop a bias towards certain parts of the original data set [9].

Test set

The test set is used to evaluate the performance of the trained network. It is important that the test set is only used to evaluate the network *after* it has been trained, and the network will not be further adjusted if it scores badly on the test set [9]. Generally, a network that scores poorly on both the training set and the test set has not been trained long enough or well enough, resulting in it underfitting to the training data. If a network scores well on the training set, but poorly on the test set, it has been trained too much, which resulted in it overfitting to the training set. Figure 3.7 shows examples of an underfitted and overfitted neural network fits. Thus, the test set is used as a measure for the generalization ability of a network [9]. Oftentimes, the test set is also used to choose the best alternative from a selection of networks with differing topologies trained for the same problem.

Validation set

A validation set is used to evaluate the performance of a network while it is being trained. Generally, the validation set is used to prevent overfitting the network to the training data [9]. This is done by checking the networks performance on the validation set a number of times during the training process, if the validation performance stops increasing, training is stopped (a process called *early-stopping*). As opposed to the training and test set, a validation set is not mandatory to use, as a network could be trained and evaluated without it. However, when data is abundant, the use of a validation set can significantly improve the process of training a network.

Data division

In general, the training set will contain the majority of all available data, as more training data will lead to a higher predicting accuracy of the final network [9]. However, the exact percentage of the original data set the training set takes up depends on the requirements the network has to fulfill and the amount of available data, but a good rule of thumb is to have a training set which contains 60-80% of the original data available [9].

When dividing the original data set into the two or three subsets, it is important for each subset to be representative of the whole [57]. If they are not, it can lead to biases in the trained network or evaluation methods. For example, when the training set contains a disproportionate amount of instances within a certain class or part of the input domain, after training it will score well for similar instances in the test set, but poorly on other types of instances. When the test set contains a disproportionate amount of a certain instance type, but the training set doesn't, the resulting trained network might have a poor evaluation score, even though in actuality it was properly trained. Similarly, when the validation set is skewed, it can lead to problems during training, be it the training process stopping too early, or going on for too long.

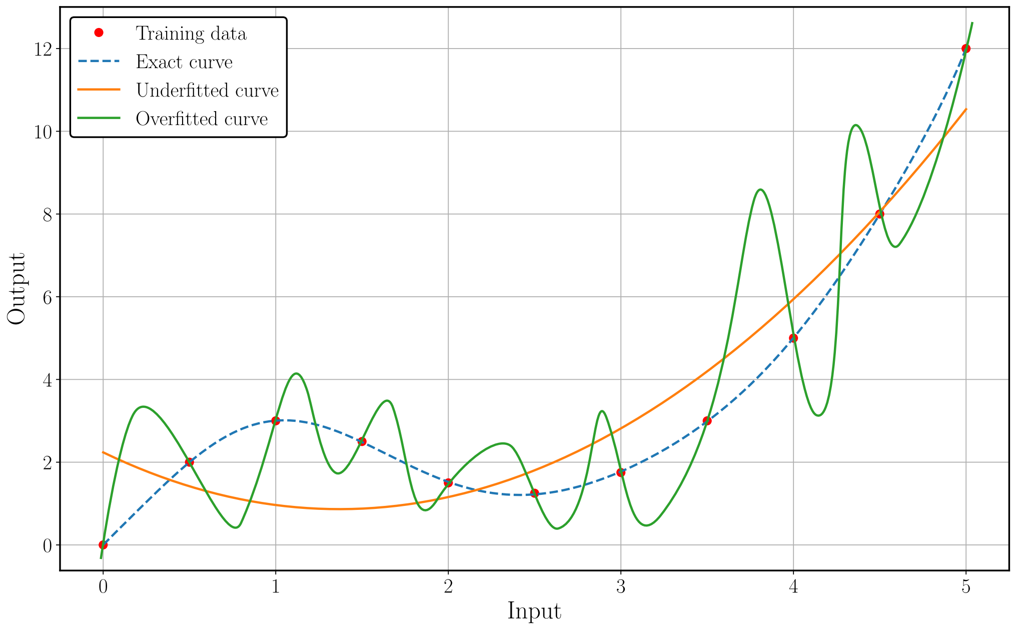


Figure 3.7: Example plot of an under- and overfitted curve. The underfitted curve (orange line) can result from the neural network having too few neurons to properly model the training data (red circles) or being given a too short training time. The overfitted curve (blue line) can result from a network that was given too many neurons, or being trained too long without proper validation.

3.3.2 Formatting

The goal of formatting is to reshape the original data set so it can be directly fed into a training algorithm. Depending on the API of the software used to train the neural network, the correct input format might differ, so the first step in formatting is determining the right format. However, most software packages require the data to be in the form of an array of horizontally concatenated column vectors, each column vector corresponding to a different input type. Oftentimes, in supervised learning, the inputs and targets are separate arrays of shape $n \times d$ and $n \times t$ respectively, where n is the number of instances (data points), d is the number of inputs, and t is the number of targets (outputs). An example of (part of) a correctly formatted data set with two inputs and one target to be used with the Keras module for Python [64], is given below.

$$\text{Input array:} \begin{pmatrix} d_1^1 & d_1^2 & d_1^3 \\ d_2^1 & d_2^2 & d_2^3 \\ \vdots & \vdots & \vdots \\ d_{n-1}^1 & d_{n-1}^2 & d_{n-1}^3 \\ d_n^1 & d_n^2 & d_n^3 \end{pmatrix} \qquad \text{Target array:} \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_{n-1} \\ t_n \end{pmatrix}$$

3.3.3 Randomizing

Randomizing of the data set is done in order to prevent problems with training a network. If the order of the training data has some underlying structure, training can either take longer, or fail, to reach a desired accuracy where a randomly ordered training set would.

Figure 3.8 gives an example of a very simple regression problem, where a neural network needs to be trained to predict the value of y for a given input x. In this case the data seems to be linearly correlated in the first section (the part inside the dashed box), before slowly transitioning to an exponential correlation. If the training set would be fed into the learning algorithm sequentially (beginning at the lowest value of x), the weights would initially be updated to fit a linear curve (the blue line), however, as soon as the learning algorithm encounters values outside the dashed box, the data stops following a linear correlation, and the accuracy of the network trained so far suddenly shows a significant drop. As a result, the learning algorithm has to readjust previously determined weights to fit with the data it is currently being fed, which might cause it to loose some of its accuracy it had on the previous parts of the data set, and reduces overall training efficiency. Thus, for every pass through the training set, the learning algorithm has to constantly keep readjusting what it has previously done, severely slowing down the learning process. This problem is exacerbated even more when using non-batch learning techniques.

In short, randomizing makes sure that the training data (or batches of it) are representative of the entire data set, and not only of sections of it.

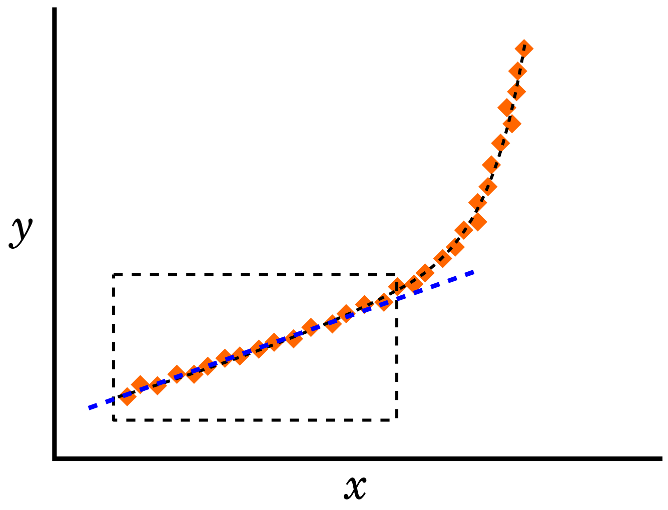


Figure 3.8: Example of a simple regression problem to help illustrate the problem of sequentially ordered data sets. If a network is trained first on the data point in the dashed box, it will result in a first fit that is linear, but as the network is trained on more data points outside the box, the learning algorithm will have to significantly readjust it's previously determined fit to the new more exponential trend, reducing its learning efficiency.

3.3.4 Feature scaling

Feature scaling refers to (reversibly) changing the values of the data sets to lie within a certain interval, or around a certain point. Two often used scaling methods are min-max scaling and standard scaling. In min-max scaling (or normalization) the data is scaled to lie in the interval [0, 1], where the minimum value of the original data set corresponds to 0, the maximum value to 1, and all other values lie in between. This is achieved by subtracting the the minimum value from the value to be scaled, and dividing by the difference between the maximum and minimum values:

$$x_i' = \frac{x_i - \max(\boldsymbol{x})}{\max(\boldsymbol{x}) - \min(\boldsymbol{x})} \tag{3.7}$$

In standard scaling (or standardization), the data is transformed to lie around the zero point with unit standard deviation. This is done by subtracting the original data sets mean (μ) from the value to be scaled, and dividing by the standard deviation (σ) of the original data set:

$$x_i' = \frac{x_i - \mu}{\sigma} \tag{3.8}$$

Feature scaling is done for a variety of reasons, including standardizing of the (often largely varying) data set, reducing the relative scales of different input types, and preventing immediate saturation of the activity function (when a bounded activation function is used).

3.4 Neural network training

This section covers the basic principles of what the process of training an artificial neural network entails, and the different algorithms involved in the training process. The coming subsections pertain to the process of supervised learning, but are relatively simple to extend to unsupervised learning applications.

3.4.1 The basic concept of training

A neural network is trained in a *data-driven* manner, meaning that it learns by example from a (preferably large) set of predetermined data instances [9,56]. The data set contains many instances of inputs with a known corresponding output, i.e. examples of the relation the neural network is intended to learn. The network is given the example inputs and predicts a corresponding output, the networks performance is compared to the real data example outputs and given a score or measure of accuracy/error, based on which the weights and biases of the network are adjusted slightly to improve the networks predictive ability, and decrease its error. After the weights are adjusted, the network is again made to predict the data examples, given a score in the form of a loss function, and the weights and biases are again updated [9]. The prediction, scoring, and weight updating steps are repeated until a set number of iterations (called *epochs*) is exceeded or a desired accuracy of the networks predictions is reached. Step-by-step, a training algorithm performs the following actions [9]:

1. *The initialization step.* The weights and biases of the neural network are initialized and an iteration loop is started.
2. *The prediction step.* The example data inputs are given as inputs to the neural network, which predicts a certain output.
3. *The scoring step.* The output predicted by the neural network is compared to the output from the example data set, and given a performance score, called the *loss function.*
4. *The optimization step.* An optimization algorithm is used to find the changes in the weights and biases needed to achieve a decrease in the value of the loss function, and thus an increase in the networks performance. More details on optimization are given in Section 3.4.2 and Appendix D.
5. *The weight-update step.* The optimization algorithm determines which weights and biases currently have the biggest influence on the loss function and adjusts them such that the loss function decreases in value.

(optional) *The validation step.* The network is scored based on a data set containing instances not found in the training set. If this score does not improve for more than a predetermined number of iterations, training is stopped.

Figure 3.9 shows a schematic overview of the steps described above, leaving out the validation step.

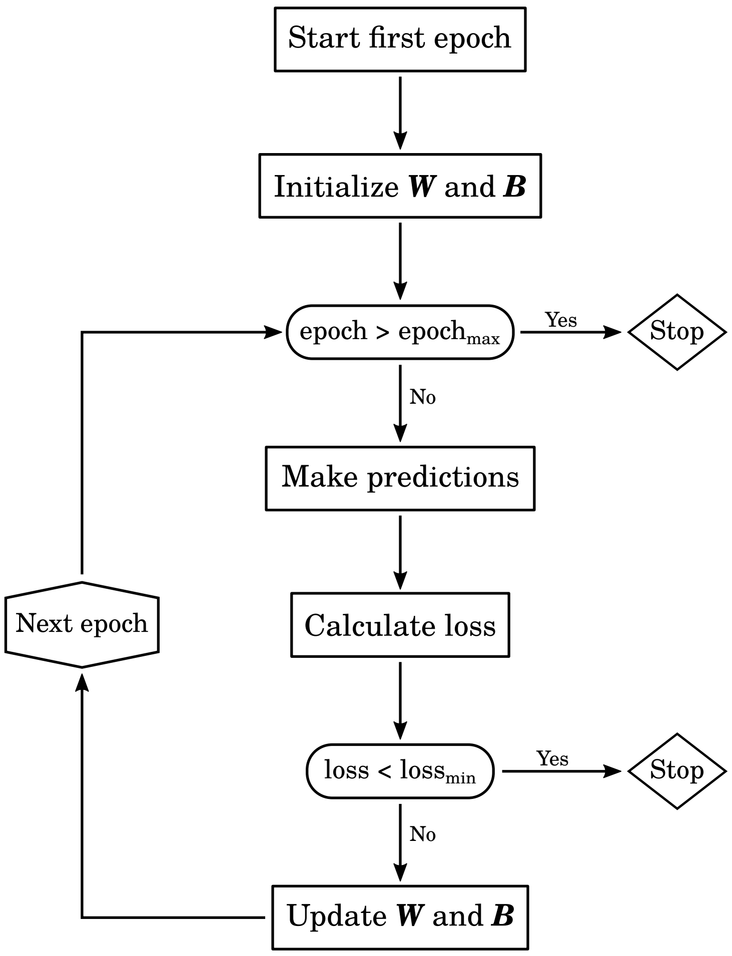

Start first epoch
Initialize $\boldsymbol{W}$ and $\boldsymbol{B}$
epoch > $\text{epoch}_{\text{max}}$
Yes
Stop
No
Make predictions
Next epoch
Calculate loss
loss < loss_{min}
Yes
Stop
No
Update $\boldsymbol{W}$ and $\boldsymbol{B}$

Figure 3.9: Schematic overview of the steps taken by a general neural network training algorithm. Here, $\boldsymbol{W}$ stands for the set/matrix of all weights of the network, and $\boldsymbol{B}$ stands for the set/matrix of all biases. $\text{epoch}_{\text{max}}$ is the maximum number of epochs, and loss_{min} the minimum desired value of the loss function specified.

The loss function

A loss function is an analytic function of the performance or error of a neural network, and is the objective function of the optimization algorithm responsible for finding the weights and biases which lead to best performing network [9].

There are a variety of loss functions that can be used as performance measures. Which loss function to use depends on many factors, but most important are the type of neural network that is trained (and how it learns) and what restrictions are imposed on the error of the network.

For example, common loss functions for a regression network (a network that predict a continuously valued output based on a continuously values input) are the mean-squared error (MSE), and mean-absolute error (MAE) [9]:

$$J_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(t_i - o_i)^2 \tag{3.9a}$$

$$J_{\text{MAE}} = \frac{1}{n}\sum_{i=1}^{n}|t_i - o_i| \tag{3.9b}$$

Here,

J	is the value of the loss function,
n	is the number of data examples,
t_i	is the target value of data instance i, and
o_i	is the value predicted by the neural network of data instance i.

As can be seen from the equations above and Figure 3.10 the MSE scales quadratically, while the MAE scales linearly. As a result, the MSE weighs large errors much more heavily than small errors, while the MAE weighs all deviations equally. Therefore, a network trained using the MSE will be less likely to make prediction that are very far off (outliers), but as it puts less emphasize on correcting small errors, the average error might be higher than for a network trained with the MAE as a loss function. Depending on what is required from the neural network, it might be better to use one loss function over the other.

An often used loss function for a classification network is the (multi-class) cross-entropy (CE), which measures the deviation between two or more probability distributions [9]:

$$J_{\text{CE}} = -\frac{1}{n}\sum_{i=1}^{n}[t_i \log o_i + (1 - t_i)\log(1 - o_i)] \tag{3.10}$$

Other loss functions include: log-likelihood, expectation loss, Chebyshev loss, hinge loss, and Tanimoto loss [65].

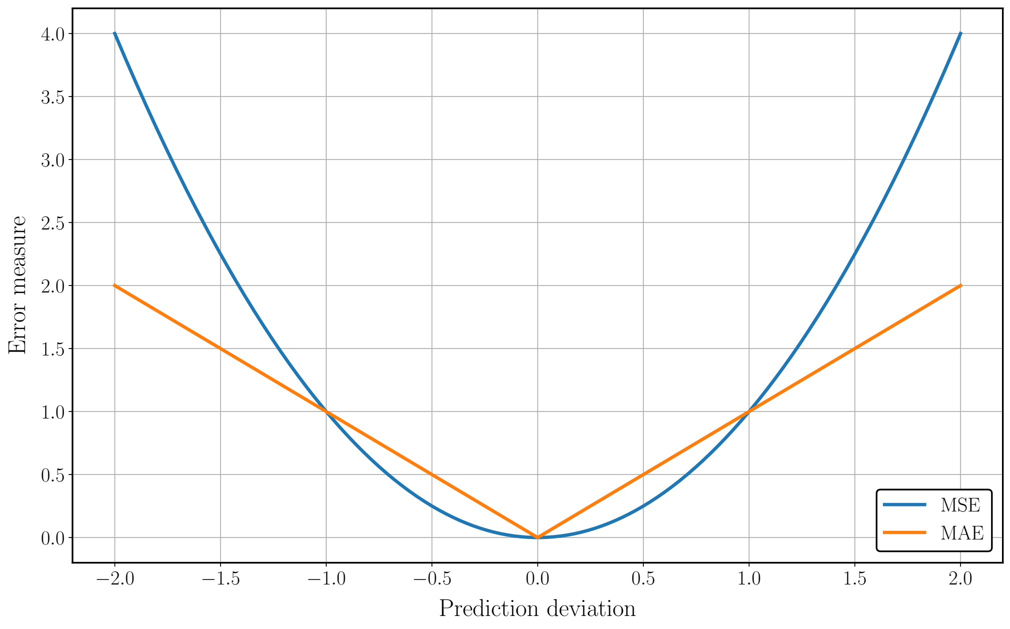


Figure 3.10: Comparison between the mean squared error (MSE) loss function (blue line) and mean absolute error (MAE) loss function (orange line). The plot shows that for low absolute prediction deviations, the MAE error function has a higher value, while for large absolute deviation, the MSE function has a higher error. Thus, the MAE more strictly "punishes" values close to correct value, resulting in a network with generally better average deviation, while the MSE more strictly punishes predictions that are very far off, leading to a network with less pronounced outliers.

Batch and incremental training

In *batch* training, the neural network is trained on the entire training set at once. Predictions are made on every training example and their errors are pooled together into a single loss factor, based on which the weights and biases are updated [66].

As the entire data set is processed by the algorithm simultaneously, a lot of computing power and memory is demanded, and the larger the data set, the larger this demand. Consequently, for very large data sets the required computing power is too high for most run-of-the-mill home computers, and one must resort to the use of super computers or an alternative learning method, such as *incremental* learning.

Contrary to a batch learning algorithm, an *incremental* learning algorithm does not process the entire training set in one go, but in multiple parts. The neural network is given a (random) selection of training instances, called a *mini-batch* or just *batch*, from the whole data set. For each batch, the loss value is determined, based on which the weights and biases of the network are updated [66]. After one batch is done, the algorithm moves on to the next batch until all have been processed, marking the end of the training epoch. The best size of the batches is usually determined by the user through empirical rules or trial-and-error.

Incremental learning can prevent the problem of limited computational memory, and can in addition be beneficial in the prevention of certain biases [66]. On the other hand, incremental training algorithms are often slower due to the fact that the weights and biases of the network are updated multiple times during each epoch (once for every batch).

3.4.2 Optimization

A neural networks loss function gives an analytic expression of the performance of the network, the lower the value of the loss function, the higher the performance of the network. As the output of a neural network is a function of its weights and biases, and the loss function is based on the networks output, the loss function is therefore also a function of the networks weights and biases. It follows that, by changing the weights and biases of the network, it is possible to influence the value of the loss function. Moreover, a set of weights and biases must exist which results in the lowest value of the loss function possible.

The process of adjusting a functions input variables in order to find the functions minimum (or maximum) output is called *optimization*, and is often done numerically using an *optimization algorithm* [40, 67, 68].

The basic optimization algorithm structure

An optimization algorithm is iterative, meaning it repeats a set of calculation step in order to reach a desired solution. The steps taken by a general optimization algorithm are [40, 67, 68]:

1. *The initialization step.* The algorithm is given a set of starting input values (initial guess) $\boldsymbol{x}^0$, a desired convergence accuracy conv_{min} is defined, a maximum number of iterations iter_{max} is specified, and an iteration loop is started.

2. *The differentiation step.* The output of the objective function $y^k = f_\text{o}(\boldsymbol{x}^k)$ for the current value of $\boldsymbol{x}^k$ is calculated, and the gradients $\nabla \boldsymbol{x}^k$ of the objective function at the point $\boldsymbol{x}^k$ are found by numerical differentiation.

3. *The update step.* Based on the gradient calculated in the previous steps and a predefined update scheme, the current set of input values $\boldsymbol{x}^k$ is updated: $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha\Delta\boldsymbol{x}^k$. Where $\Delta\boldsymbol{x}^k$ is called the *direction vector*, and α the update step size. If required, an optimal value of α can first be found using a line-search algorithm [68].

4. *The convergence step.* The convergence of the solution $\boldsymbol{x}^{k+1}$ is checked and compared with the desired convergence. If the current convergence accuracy is lower than (or equal to) the desired convergence accuracy, iterations are stopped, if not, the algorithm goes back to step 2 and replaces the old value of $\boldsymbol{x}^k$ with the new value $\boldsymbol{x}^{k+1}$. The algorithm also stops if the maximum number of iterations is exceeded.

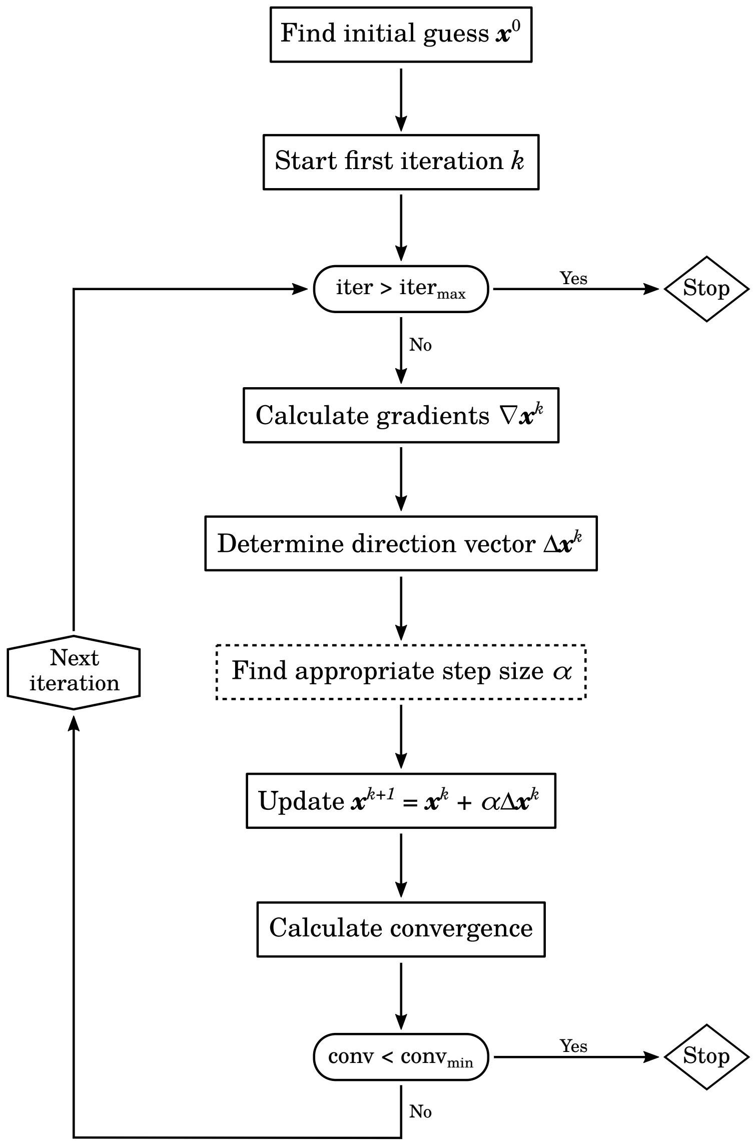

Find initial guess $\boldsymbol{x}^0$
Start first iteration k
iter > iter_{max}
Yes
Stop
No
Calculate gradients $\nabla \boldsymbol{x}^k$
Determine direction vector $\Delta \boldsymbol{x}^k$
Find appropriate step size α
Update $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha \Delta \boldsymbol{x}^k$
Calculate convergence
conv < conv_{min}
Yes
Stop
No
Next iteration

Figure 3.11: Schematic overview of the steps taken by a basic optimization algorithm.

Figure 3.11 gives a schematic overview of the steps described above. The scheme described here is an example of the most basic *first order* optimization algorithm, such as the *gradient descent* algorithm [40]. Other optimization algorithms might include additional steps to find the gradients of the objective function or calculate a more effective direction vector. When applied to neural networks, the objective function of the optimization algorithm is the loss function, while $\boldsymbol{x}$ corresponds to the values of the weights and biases of the network.

The algorithm described above is *deterministic*, which means that if all settings are kept constant, it will always find the same solution. On the other hand, many optimization algorithm include some form of randomness in their calculation methods in order to prevent getting stuck in local minima, and more effectively go through the search-space. Appendix D gives some more details on the difference between deterministic and stochastic optimization algorithms.

3.4.3 Backpropagation

Backpropagation is an algorithm that makes it easier (less computationally intensive) to calculate the gradients of a neural networks loss function with respect to all the weights and biases in the network [69]. Backpropagation was first described in 1986 by Rumelhart, Hinton, and Williams [70], and works by propagating the final error of the network backwards through the network in order to determine to current contribution of each weight and bias of the network to the networks predictive error.

Once these contributions are known, any gradient-based optimization algorithm can be used to update the weights and biases in order to minimize the value of the loss function. It should thus be noted that backpropagation is not an optimization algorithm in of itself, but merely a method for quickly calculating gradients in a large network [69]. In fact, backpropagation is a special case of a much broader differentiation technique called *automatic differentiation* [69].

Backpropagation is best explained through the use of an example. For instance, given the network shown in Figure 3.12, calculate what effect a change in weight w_α has on the value of the loss function J, i.e. determine the (partial) derivative of the loss function J with respect to weight w_α. To do so, a backpropagation algorithm takes the following two steps [53, 69]:

1. *Forward propagation.* Given an input instance, the algorithm propagates information forwards through the network and calculates the activation values of all neurons in the network, as well as their partial derivatives, and the value of the loss function (error).

2. *Backwards propagation.* The error is propagated backwards through the network to determine how the activations of neurons at the end of the network are affected by weights and biases that come before them.

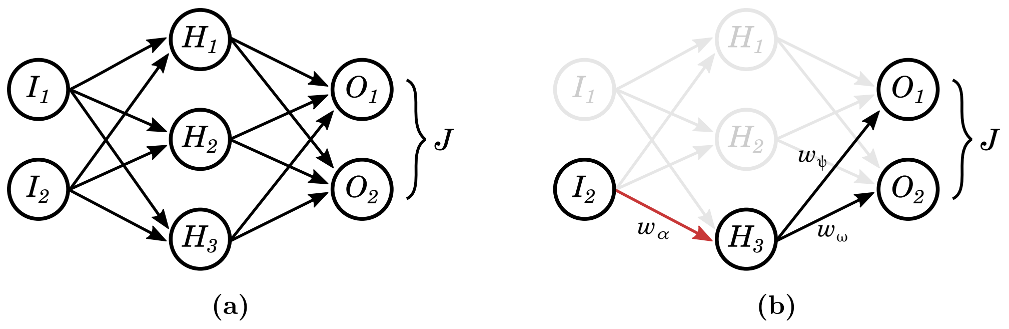


Figure 3.12: Example network used to explain backpropagation. Complete network (a), and the network including only the parts of the network that directly depend on the value of w_α (b). Here, J stands for the value of the loss function.

For the example at hand, the first step comes down to determining which neurons in the network are a function of the weight w_α and calculating their derivatives. For example, for the network in Figure 3.12, you find that w_α affects H_3, which in turn affects both O_1 and O_2, which both affect the value of the loss function J. In mathematical terms, the following relations are found:

$$H_3 = f(w_\alpha) \tag{3.11a}$$

$$O_1 = g(H_3) = g(f(w_\alpha)) \tag{3.11b}$$

$$O_2 = g(H_3) = g(f(w_\alpha)) \tag{3.11c}$$

$$J = h(O_1, O_2) = h(g(f(w_\alpha)), g(f(w_\alpha))) \tag{3.11d}$$

As can be seen, J will be affected by changes in the value of O_1, which will itself be affected by changes in H_3, which will be influenced by changes in the value of w_α. Therefore, to determine the change in the loss function J as a result of changes in w_α, it is necessary to take into account both the effect the changes in O_1 and O_2 have on J itself, as well as the effect a change in w_α has on both O_1, and O_2. In the same vein, to know how a change in w_α affects O_1, one must take into account both the effect H_3 has on O_1, and the effect w_α has on H_3, etc. This is more or less what is done in the second step of the backpropagation algorithm.

In mathematical terms the backwards propagation step is just a repeated application of the chain rule of calculus, which states that if $z = f(y)$, and $y = g(x)$, than the derivative of z with respect to x is [69]:

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x} \tag{3.12}$$

Which, when applied to the example at hand, gives:

$$\frac{\partial J}{\partial w_\alpha} = \frac{\partial J}{\partial O_1}\frac{\partial O_1}{\partial w_\alpha} + \frac{\partial J}{\partial O_2}\frac{\partial O_2}{\partial w_\alpha} \tag{3.13a}$$

$$\frac{\partial O_1}{\partial w_\alpha} = \frac{\partial O_1}{\partial H_3}\frac{\partial H_3}{\partial w_\alpha} \tag{3.13b}$$

$$\frac{\partial O_2}{\partial w_\alpha} = \frac{\partial O_2}{\partial H_3}\frac{\partial H_3}{\partial w_\alpha} \tag{3.13c}$$

The directions of the forwards and backwards steps of the backpropagation algorithm should be clear from the example given above. In the *forwards* step you start at the beginning of the network with the value of the weight w_α, and determine which neurons in the remaining layers of the network are a function of w_α. Additionally, the forwards step is also used to determine and store the partial derivatives of each neurons activation with respect to the weights and biases connected to it, so that is only has to be done once, instead of repeatedly in the backwards step.

In the *backwards* step, you start at the end of the network with the value of the loss function, and check how its derivative(s) depend on derivatives of the activations of neurons that come before it. As the derivatives were already calculated in the forwards step, one can just apply the chain rule instead of having to calculate the entire partial derivative of J with respect to every weight and bias in the network. This saves tremendously on computing time [69].

In the example above, the focus was solely on the weight w_α, and all parts of the network which were not a function of the value of w_α, were disregarded. This not only included all parts of the network not part of the path from I_2 to J, but also the values of I_2 itself and the weights w_ψ and w_ω, which are all three parts of the path of influence of w_α. However, while they affect the value of J, their values themselves do not depend on the value of

w_α, therefore they were not included in the expressions of Equation 3.11 and Equation 3.13. However, once the full expressions for the derivatives are evaluated, the values of I_2, w_ψ and w_ω will have to be taken into consideration.

3.5 Applications of neural networks to fluid phase equilibria

In this section, a number of examples of other studies that investigated the application of artificial neural networks to phase stability, equilibrium calculations, and fluid property predictions are given.

3.5.1 Phase stability analysis

To the best of the authors knowledge, not much research has been done on the problem of solving phase stability analysis using artificial neural networks. The studies that have been found are briefly discussed here.

In their 2006 paper, Schmitz et al. [71] investigated the use of ANN in determining the phase stability (number of phases present) of an ethanol–ethyl acetate–water mixture. They did so by training neural networks on the bubble and dew point lines of the mixture over the entire range of mole fractions and a temperature range of around 40 K. They used data generated through conventional methods. Their trained neural networks achieved accuracies of over 99.9%.

Kashinath et al. [5] developed a hybrid ANN algorithm that calculates the phase behavior (stability and phase split) of a mixture of hydrocarbons. They used a support vector machine [9] to determine the probability of the mixture being in a certain phase regime, if the probability is not higher than a user-defined cut-off, the prediction is rejected and the algorithm falls back on a negative flash calculation [21] in order to determine phase stability. By adjusting the the probability cut-off parameter, one can trade off accuracy for calculation speed. In addition, their algorithm also uses an ANN to predict the values of the mixtures K-values, again, if this prediction is not sufficient enough, the algorithm reverts back to conventional calculation methods, using the K-value prediction as an initial guess.

Gaganis et al. [72] did investigate the use of machine learning in solving the phase stability problem, but they only used support vector machines, and not ANN.

3.5.2 Equilibrium calculations

Contrary to phase stability analysis, a lot more research has been done on equilibrium calculations, a few studies are highlighted and briefly discussed here.

Farzi and Nehad [73] used a simple MLP type neural network to predict the vapor-liquid compositions for 11 binary mixtures containing acetone. They predicted the liquid mole fraction (x) and vapor mole fraction (y) of the acetone based on experimental values of critical temperature, critical pressure, acentric factor, and temperature and pressure. They achieved average percentage errors of around 1% for most of the mixtures considered.

In a similar vein, Vaferi et al. [74] predicted the bubble pressure and vapor composition of 9 different binary mixtures containing ethanol using a two-layer ANN based on inputs including the critical temperature, critical pressure, acentric factor, boiling temperature, temperature, and liquid phase mole fraction. In their work, the neural networks were trained on experimental data and compared to the Peng-Robinson equation of state; they concluded that the ANN had superior predictive capabilities.

Moghadam and Asgharzadeh [75] used group method of data handling [76] type neural network to predict the liquid-liquid equilibrium compositions of three ternary mixtures, they reported mean relative error percentages of around 2-5% at constant temperature, and 5-13% at variable temperature.

Additional applications of ANN to equilibrium predictions can be found in [77–80].

3.5.3 Property value prediction

When it comes to the application of neural networks to parts of the classic flash problem, by far the most research has been done in the area of property value predictions. Three of these studies are described here.

Chouai [81] et al. used an MLP network to predict the vapor and liquid compressibility factors Z for three different refrigerants, using a temperature range from 240 to 340, and a pressure range up to 20 MPa. They trained the neural networks on the value of the compressibility factor Z, which they then used to calculated enthalpy H and entropy S, achieving a maximum error of 27 J/mol and 0.5 J/mol/K respectively. Similar as to the approach in this text, they used conventional methods (REFPROP 6.0) to generate property data to train on and compare to.

Similarly, Mora et al. [82] used ANN to predict vapor and liquid volume/density, enthalpy, and entropy of a number of refrigerants based on experimental data, achieving very high accuracy.

A hydbrid method was developed by Seifi and Abedi [42], who used an ANN to predict an initial guess to be used in a conventional bubble point pressure algorithm, instead of directly predicting the bubble point pressure using ANN.

Finally, Valderrama et al. used a combination of ANN and Group Contribution theory to predict the density and melting temperature of ionic liquids [83, 84]. They reached error scores of lower than 1% for density and a test set absolute percentage deviation for temperature of around 15%.

Additional applications of neural network to property predictions of different types of systems (including hydrocarbons, petroleum fluids, ionic liquid, alcohols, pharmaceuticals, and even foodstuffs), most using similar approaches as the studies described here, can be found in [85–99].

3.5.4 What this work adds

In the previous sections, it was shown that using ANN to predict phase stability, equilibrium compositions, and property values is not a completely new field of study. This work aims to add to the field in the following ways:

- Almost all of the studies cited in the previous sections focussed either on phase stability or property prediction only. Only Kashinath et al. [5] considered both. In this work, phase stability and property prediction were combined into a single algorithm in order to investigate how the former affects the latter, and to include all parts of the classic flash problem into one neural network based algorithm.

- Most of the works cited focussed either on pure compounds or mixtures with constant or a limited number of different compositions. When an entire range of compositions was considered (such as in [5, 71]), other variables such as pressure or temperature were often fixed. In this study, none of the input variables to the flash problem have been kept constant, so that the resulting algorithm would be applicable to a wide range of interest.

- The works cited above consider only pressure and temperature (plus other properties) as inputs to the neural networks. In this study, combinations of pressure and entropy, entropy and volume, and enthalpy and volume are also considered. In addition, most of the above cited works focussed on a limited number of output properties, while in this project all thermodynamic variables and energy functions are included.

Chapter 4

Algorithm design and implementation

In this chapter the design and implementation of the neural network based flash algorithm are described. The first section will outline the solution method developed and how it was implemented in the Python programming language. The second section will briefly cover the data generation method, and provide all relevant details of the neural network training parameters.

4.1 Neural network flash algorithm design

4.1.1 Algorithm requirements

In this section the properties the neural network flash algorithm must be able to predict are given. Before the specific details are outlined, the design of the simulation engine the algorithm will be a part of will be briefly explained. Not only will this give a better idea of the context in which the algorithm will be operational, but the design of the simulation engine will directly influence which properties are required to be predicted.

Simulation engine design

The proposed simulation engine keeps the following order of operations:

1. System inputs, such as initial temperatures, pressures, compositions, geometry, etc., are specified by the user.

2. The system inputs are used by the PT-flash to initialize all relevant thermodynamic properties.

3. An HV-flash is used to determine changes in pressure and entropy as a result of changes in enthalpy (due to mixing, reactions, or heat transfer).

4. A PS-flash is performed to determine the maximum and minimum volumes in the system based on the maximum and minimum pressures calculated with the HV-flash.

5. An SV-flash is executed repeatedly to iterate on the volume flows in the system.

6. After the work in the system is internally resolved with the SV-flash, the results are fed back into the HV-flash and the next time step is started. Steps 3-5 are repeated until a specified simulation duration is reached.

7. At a predetermined interval of time steps (defined through the system inputs), system outputs, such as temperatures, compositions, pressures, mass flows, etc. within the system, are given back to the user or saved to be reviewed at a later time.

These steps are represented schematically in Figure 4.1. In this configuration, the PT-flash is executed only once for every simulation, the HV-flash and PS-flash are executed at every time step, and the SV-flash is repeated multiple times at every time step. Thus, the neural network flash algorithm needs to include four flash types: a PT-flash, a PS-flash, an SV-flash, and an HV-flash.

The steps detailed above are part of the *thermodynamic backbone* of the simulation engine, effects such as heat transfer, mixing of components, and chemical reactions are taken care of by a different part of the simulation engine, and will not be discussed in this text.

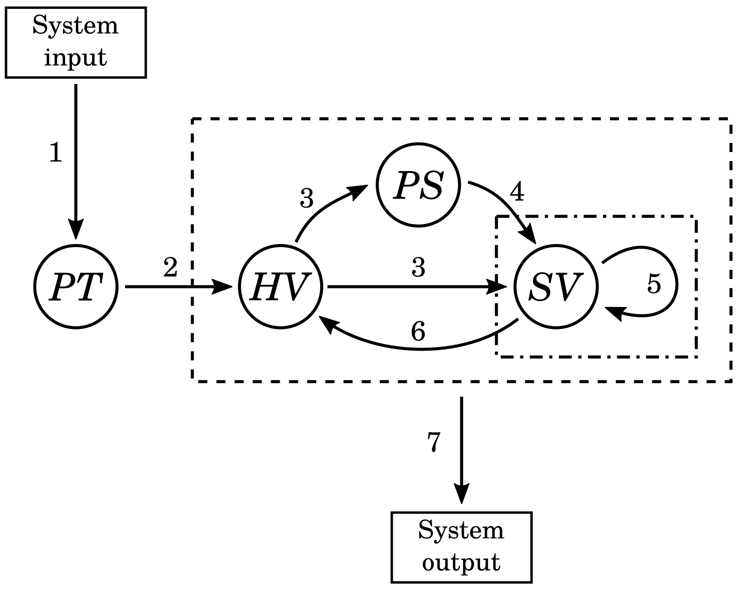


Figure 4.1: Schematic overview of the order of the four required flashes. The flashes within the dashed box are executed every time step, the flash within the dash-dotted box is executed multiple times per time step. The numbers indicate the order of execution.

Property requirements

Based on the order of execution described in the previous section, the inputs and outputs of each flash calculation can be determined. For example, as the results from the PT-flash will directly be used as inputs to the HV-flash, the PT-flash must *at least* be able to predict enthalpy and volume. Similarly, all other flash types have certain essential properties they must be able to predict. For the sake of flexibility, the decision was made to also include additional non-fundamental properties in each flash, which depend on the flash type, but include for all types internal energy (U), Gibbs free energy (G), and Helmholtz free energy (A).

Vapor-liquid phase compositions are calculated based on predicted values of what are dubbed here as *component vapor fractions* (β_i). In early design stages phase compositions were calculated based on the total vapor fraction β and values of the distribution coefficients K_i, using the Rachford-Rice equations to predict vapor and liquid compositions x_i and y_i. However, since all values were predicted independently it was found that inaccuracies in the separate predictions led to even higher inaccuracies in the ultimate values of x_i and y_i, often leading to mass balance violations.

In order to be able to more easily insure mass balance the decision was made to use individual component vapor fractions to calculate the vapor phase composition in absolute moles, and subtracting the results from the total feed mole numbers to get the liquid phase mole numbers and composition. In addition, this also reduced the number of independently predicted variables from three to two, potentially increasing accuracy of the two-phase composition predictions. The component vapor fractions are defined as:

$$\beta_i = \frac{n_i^{\mathrm{V}}}{n_i^{\mathrm{F}}} \tag{4.1}$$

And can be used to calculate phase compositions as follows:

$$n_i^{\mathrm{V}} = \beta_i n_i^{\mathrm{F}} \tag{4.2a}$$

$$n_i^{\mathrm{L}} = n_i^{\mathrm{F}} - n_i^{\mathrm{V}} \tag{4.2b}$$

$$y_i = \frac{n_i^{\mathrm{V}}}{\sum_{i=1}^{c} n_i^{\mathrm{V}}} \tag{4.2c}$$

$$x_i = \frac{n_i^{\mathrm{L}}}{\sum_{i=1}^{c} n_i^{\mathrm{L}}} \tag{4.2d}$$

Here,

β_i	is the vapor fraction of component i in [mol/mol],
n_i	is the number of moles of component i in [mol],
x_i	is the mole fraction of component i in the liquid phase in [mol/mol],
y_i	is the mole fraction of component i in the vapor phase in [mol/mol],
c	is the total number of components in the mixture, and
V, L, and F	are superscripts indicating the vapor phase, liquid phase, and original feed respectively.

The situation where only mole fractions (z_i) are provided can be considered a special case in which the total amount of moles is equal to one; the above equations remain valid.

Table 4.1 gives an overview of all essential and additional properties of each flash type.

Table 4.1: Summary of all essential and additional inputs and outputs of each flash. The input mole numbers correspond to the total feed amounts (n^{F}).

Flash type	Inputs	Essential outputs	Additional outputs
PT	P, T, n_i	H, V, β_i	S, U, A, G
PS	P, S, n_i	V, β_i	T, H, U, A, G
SV	S, V, n_i	H, P, β_i	T, U, A, G
HV	H, V, n_i	S, P, β_i	T, U, A, G

It should be mentioned that all properties to be predicted are in their intensive form (on a per mole basis) so that the predictions can be generalized to systems of any size. As long as the absolute mole numbers of a given mixture are specified, input values can also be given in extensive form, which can be made intensive by dividing by the total amount of moles present.

Density is calculated based on the predicted value for the volume, as well as the average molar weight of the mixture of components. Heat capacity, viscosity, and thermal conductivity are currently calculated using standard and relatively simple temperature correlations (the same as used in TEA [11], see Appendix C), but can in the future be replaced with neural networks if the need is there.

During development the idea arose to predict all thermodynamic properties per flash based on the ideas of fundamental equations: predicting one of the four energy functions described in Section 2.1.3 and 2.1.4 and using numerical derivatives to calculate the values of all other properties. This was an attractive option as is would require predicting much less different properties, reducing the total number of neural networks required to be trained, and making it possible to focus all effort on improving only one property. In addition, it would also improve thermodynamic consistency for each individual flash.

However, during early stages it was concluded that due to inaccuracies in predictions of the values of the fundamental equations, the resulting numerical derivatives were too inconsistent, leading to inaccurate and sometimes even un-physical values of the other properties. Therefore, the approach was not pursued any further.

Components included

The original aim of this project was to include mixtures of up to seven different components. However, it was quickly concluded that this required an unreasonable amount of data, unless a more sophisticated data generation algorithm was developed. While a concept for such an algorithm was ideated, the actual development of such an algorithm was considered to be too time-consuming and out of the scope of this work.

For this reason, the decision was made to constrain the current work to only two components: water and methanol. As briefly mentioned in the introduction, these components were chosen as they formed the smallest set of compounds that was present in one of the subsystems of the ZEF methanol plant.

It should be mentioned that the algorithm described in this thesis was developed while keeping in mind future extension to additional components, such that if the data generation problem is solved, all other parts of the algorithms and neural network training processes won't have to be adapted much.

4.1.2 Algorithm structure

Every algorithm that takes a given input and returns a corresponding output has an underlying structure that determines what actions are taken in which order to get from input to output. That structure determines the number of artificial neural networks required, the way they are coupled in order to solve flash problems, their relative order of execution, and any additional functions that may be required. In this section the implemented structure is described. Two other approaches were considered, but ultimately not chosen due to certain limitations they were considered to have. These two approaches are described in Appendix E.

Algorithm description

The neural network based flash algorithm structurally resembles the conventional flash problem approach: first a given systems phase is determined, after which property values are predicted. The first part is approached as a classification problem, in which a neural network is used to distinguish between three different classes: the liquid region, the vapor region, and the two-phase (vapor-liquid equilibrium) region. The second part is approached as a regression problem, in which different neural networks are used for each phase region to predict continuously valued properties. The decision was made to predict each property, for each phase regime, with its own network. This was done so that the amount of redundancy in a network could be reduced, i.e. if the only current property of interest is enthalpy, but the network outputs enthalpy, volume, entropy, Gibbs free energy, etc., a big part of the network will be executed that is currently not relevant, reducing computational efficiency.

Inputs to the algorithm are: (i) the values of the two thermodynamic properties which determine the systems state, i.e. value combinations of pressure and temperature, pressure and entropy, entropy and volume, and entropy and enthalpy; (ii) the composition of the water-methanol mixture; and (iii) the desired output property. As neural networks can process multiple input combinations at once, the algorithm allows the user to specify a list or array of input combinations for which property values need to be predicted.

In order to get from the specified inputs to the desired output, the algorithm takes the following steps:

1. The algorithm checks whether the desired output property specified by the user was incorporated in the flash, if not, the algorithm will raise an error. In addition, the algorithm checks whether the number of components in the input composition equals the number of components included in the flash, again, if they do not match, the algorithm raises an error.

2. If composition is given in absolute mole numbers, they are converted into relative mole fractions. If thermodynamic properties are given as extensive values, they are converted to their intensive values. Both conversions are done by dividing by the total number of moles, so if both are already given in intensive form, they are unchanged.

3. The values of the two thermodynamic properties are normalized to lie within the interval [0, 1]. See Section 3.3.4 (Equation 3.7 to be specific) for more details.

4. Depending on how the inputs were formatted, they are reformatted into a single column array which serves as the input to the neural network. See Section 3.3.2 for more details.

5. The phase classification neural network is used to determine the phase of the given input combinations. The output of phase classification is a numerical label corresponding to one of the three classes: 0 for liquid, 1 for vapor, and 2 for two-phase.

   - If the desired property was phase classification, the output array from the classification neural network is returned.

6. After phase classification, the input array is indexed and divided into three separate arrays, one for each phase region. Each array is given as an input to the property network of the desired property corresponding to the correct phase region, and the value of the desired property is predicted.

7. The property prediction networks return scaled values, so they are rescaled to the right magnitude.

8. The output arrays of each phase region are recombined into one array based on their index, such that the order of all instances is the same after the separation and prediction process as it was before it.

9. The total output array is returned to the user.

A simplified schematic of the algorithm structure is shown in Figure 4.2 for an example case in which enthalpy is the desired property and the given mixture belongs to the liquid phase.

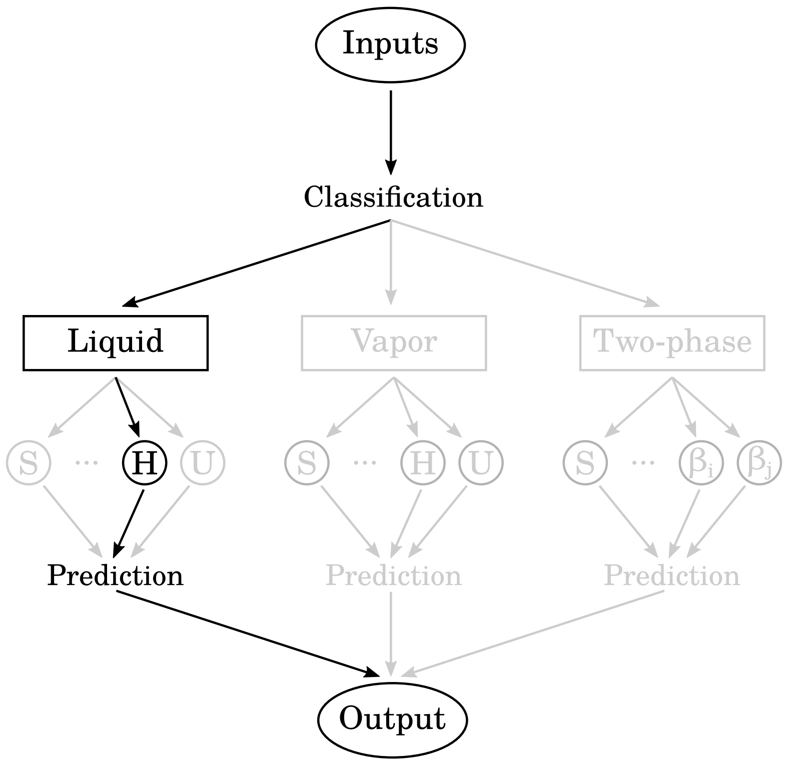


Figure 4.2: Structure of the overall neural network based flash algorithms for an example input combination belonging to the liquid phase region predicting the desired property of enthalpy (H). The algorithm first uses a classification network to determine the phase region of the inputs, and depending on the phase region and desired property chooses the appropriate regression network to predict an output value. Auxiliary functions such as scaling, indexing, splitting, and rescaling are not shown.

Implementation

The structure described above was implemented into Python using a hierarchy of four types of classes, which are shown in in Figure 4.3.

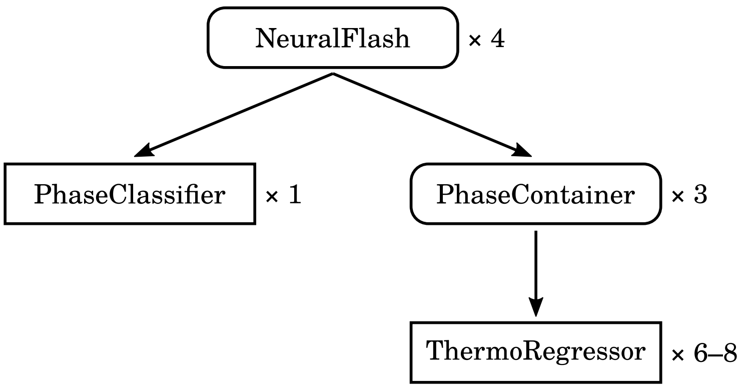


Figure 4.3: Schematic overview of the class structure implemented in Python. Each flash type is an instance of the *NeuralFlash* class which contains 1 *PhaseClassifier* instance, and 3 *PhaseContainer* instances which in turn contain 6 (liquid and vapor regions) or 8 (two-phase region) *ThermoRegressor* instances. The *NeuralFlash* and *PhaseContainer* classes do not contain any neural networks, and are only used for structural purposes.

Each class functions as follows:

1. The *NeuralFlash* class. The highest class level, the *NeuralFlash* class is used to contain all other classes, and is the sole class which the user will directly interact with. For each flash type, a different instance of the *NeuralFlash* class was constructed.

2a. The *PhaseClassifier* class. The second level *PhaseClassifier* class contains the weights and biases and other relevant attributes of the classification neural network. Each *NeuralFlash* class contains only one *PhaseClassifier* instance.

2b. The *PhaseContainer* class. The second level *PhaseContainer* class is used as a "container" for the *ThermoRegressor* classes so that they are more easy to keep track of. Each *NeuralFlash* class contains three *PhaseContainer* instances, one for each phase region.

3. The *ThermoRegressor* class. Each lowest level *ThermoRegressor* class instance contains the weights and biases and other relevant attributes for each property regression neural network. It also contains the functions which are used to scale, predict, and rescale property values. Each *PhaseContainer* instance contains either 6 (for the vapor and liquid *PhaseContainers*) or 8 (for the two-phase *PhaseContainer*) *ThermoRegressor* instances, one for each property given in Table 4.1 in Section 4.1.1.

Predictions are made by calling a function of the *NeuralFlash* class, which first calls on the *PhaseClassifier* class to determine the phase of the input, after which the *NeuralFlash* class calls upon the *ThermoRegressor* class in the correct *PhaseContainer* corresponding to the desired output property.

Advantages and disadvantages

The current approach was chosen based on a number of advantages it had over other approaches considered, these include:

- Neural networks have difficulty predicting discontinuous functions, as the transition between the liquid, vapor, and two-phase regions are often (close to) discontinuous, a neural network will be inaccurate at boundaries between them. By splitting the entire input domain into the separate liquid, vapor, and two-phase regions, the phase boundary discontinuity can be avoided.

- Phase classification gives an explicit indication of which phase an input combination belongs to, this can be useful for certain applications.

- The fact that each property is predicted using its own neural network makes it possible to easily identify which properties are most difficult to accurately predict. It's therefore also much easier to improve individual properties. On the other hand, if one network is used to predict several properties, but only one needs improvement, the entire network will have to be changed and retrained, and since training is a stochastic process, it might result in worse performance for the properties that were already good.

- Similarly, the current framework allows for additional property prediction neural networks to be trained and added to the flash algorithm without the need for redesigning and retraining already existing networks.

Naturally, no approach is ever perfect, and will always have some downsides. Some of the main disadvantages of the current approach that can be expected to be present are:

- The coupling between phase classification and property prediction can lead to discrepancies. If the phase classification network incorrectly predicts a certain phase, the algorithm will use a wrong network to predict the property value, leading to inaccurate results. This problem is illustrated in Figure 4.4, where an incorrect prediction in the phase boundary (left vertical line), leads to the two-phase network being used instead of the liquid network, resulting in a significant error in the prediction of pressure.
- When many properties are desired at once, each property has to be predicted separately, which will increase total computation time. However, the benefits of using a single network per property are believed to outweigh this downside.

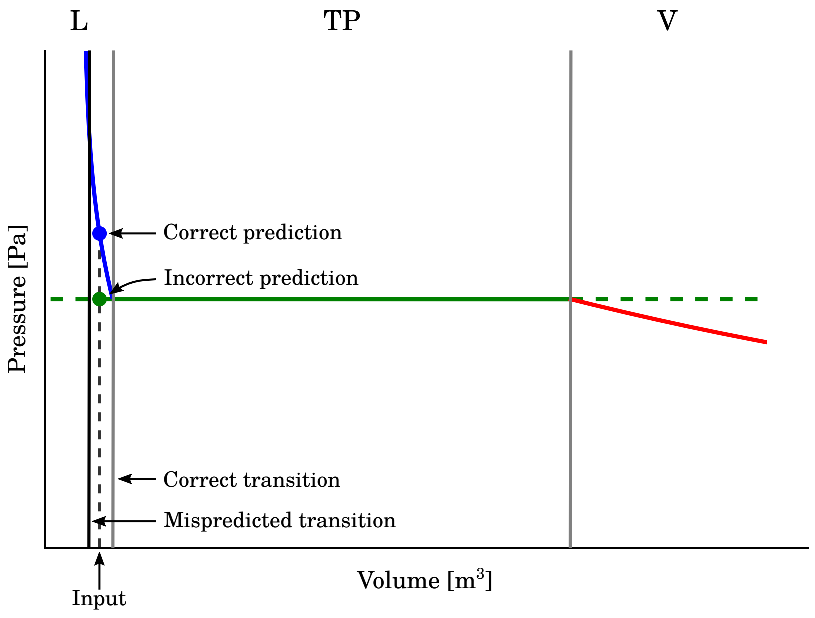


Figure 4.4: Example of incorrect output predictions as a result of incorrectly predicted phase regime. The grey vertical line is the correct boundary between the liquid and two-phase regime, if the network instead incorrectly predicts the solid vertical black line, the output will differ a lot for the same input.

4.2 Data generation and network training

In this section the methods used to generate training data are explained, and the settings used during the neural network training process are provided.

4.2.1 Data generation

As mentioned in the introduction to this thesis, data was generated using the Python plugin of the open-source Thermodynamics for Engineering Applications (TEA) property calculator [11] of the CAPE-OPEN to CAPE-OPEN (COCO) process simulator [12].

Data was generated using TEAs *PT*-flash over a range of pressures, temperatures, and mole fractions [11,31]. For each of the three inputs an upper and lower bound to the range of interest were specified, as well as the resolution of the desired range, i.e. how many points between the upper and lower bounds will be generated.

Table 4.2 shows the bounds and resolutions of the pressure, temperature, and mole fraction ranges used to generate all fluid property data. The lower bounds were based on expectations of the lowest pressure and temperature that could be encountered in the ZEF methanol plant, upper bounds were determined by looking at which values would be required to include the entire vapor-liquid envelope of the water-methanol mixtures, including some padding in both directions.

As lower pressures were of more interest to ZEF, pressure was decided to be logarithmically distributed so that more data points were present at the lower pressures of interest. Moreover, due to the large pressure range, using a linear spacing would mean that for the first two data points pressure would go from $1{\times}10^4$ to $7{\times}10^4$, skipping the majority of an entire decade, while the final values go from $2.99{\times}10^7$ to $3{\times}10^7$, which is considered unnecessarily detailed for the current investigation. On the other hand, by using a log-spacing, values go from $1{\times}10^4$ to $1.02{\times}10^4$ at the start and from $2.95{\times}10^7$ to $3{\times}10^7$ at the end.

The mole fraction resolution was chosen to be 101 so that a nice even distribution between zero and one with a step size of exactly 0.01 [mol/mol] would be generated. The resolutions of pressure and temperature were not chosen for any particular reason, only so that a large amount of training data would be available.

Table 4.2: Lower bounds, upper bounds, and resolutions of the input variables value ranges used to generate fluid property training data.

Input	Lower bound	Upper bound	Resolution	Distribution
Pressure [Pa]	$1{\times}10^4$	$3{\times}10^7$	500	logarithmic
Temperature [K]	273	700	500	linear
Mole fraction [mol/mol]	0	1	101	linear

In total, $500 \times 500 \times 101 = 25250000$ unique PTz-input combinations were generated, and for each combination the phase and values of the relevant thermodynamic properties were calculated using TEA [11]. However, for certain PTz combinations TEA could not converge to a solution, whenever this happened, the combination was left out of the total data set. Thus, the actual size of the entire data set contained 25,249,692 input combinations with corresponding thermodynamic property values. Figure 4.5 shows an example data set for a single composition of 50-50 mol% water and methanol mixture. The actual PT-classification network was trained on 101 of such data sets. Figure 4.6 shows a similar plot of pressure, temperature and output enthalpy, the regression networks would be trained on parts of this data set. Additionally, Figure 4.7 shows enthalpy data for the entire range of interest.

Figure 4.5 shows that the two-phase region makes up only a very minor part of the total range of interest. For this reason and because of the grid-based approach of data generation used, a significantly lower amount of training data was available for the two-phase region. A data generation algorithm which could solve this problem was conceptualized, but due to time constraints never implemented. At the time of development, the problem was underestimated, so no other methods were implemented to mitigate it. Section 5.2 will cover in more detail how the accuracy of the networks was affected by the fact that less data was available in the two-phase region.

It should be noted that the upper right part of Figure 4.5 is treated here as if there is a distinct boundary between the pure liquid and vapor phase regions. In reality this region is known as the supercritical region, where there is little distinction between phases and properties show diverging behavior [100]. However, in this project, the supercritical region was not incorporated as a separate phase class.

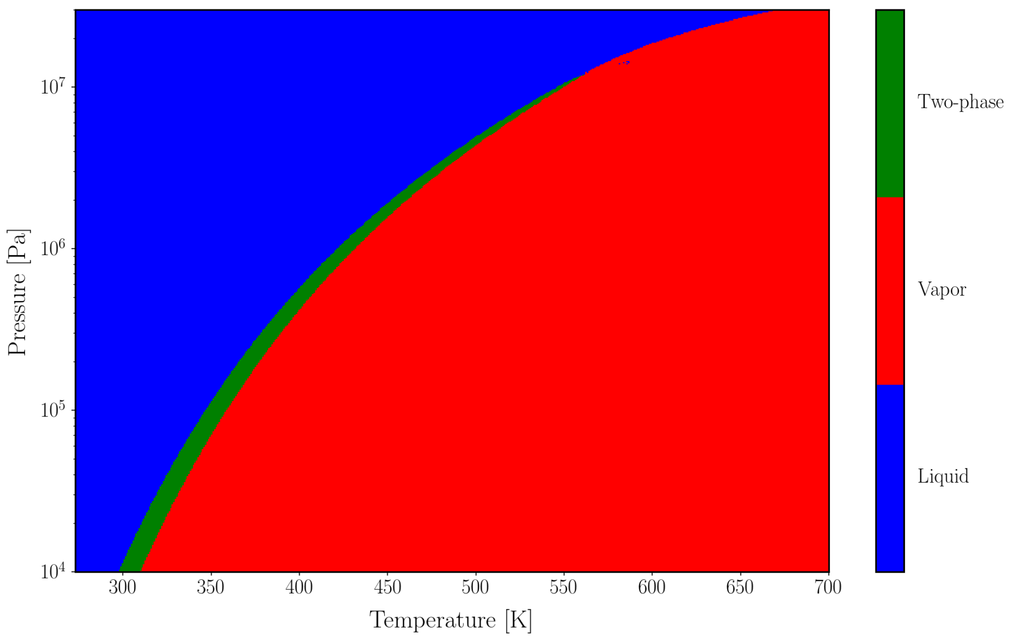


Figure 4.5: Example phase data of a 50-50 mol% mixture of water and methanol used to train a phase classification neural network.

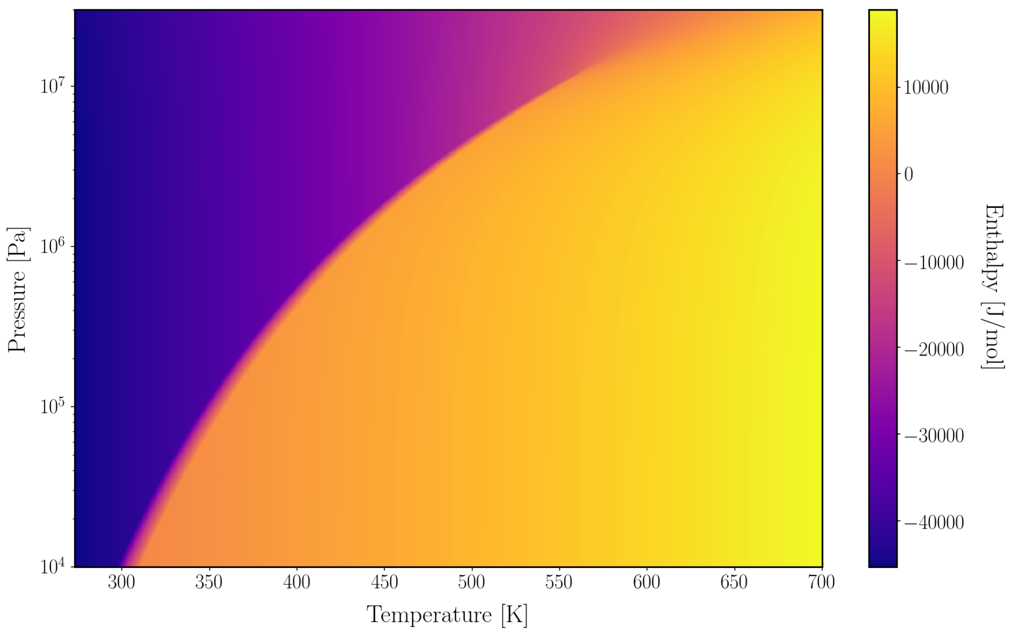


Figure 4.6: Example enthalpy data of a 50-50 mol% mixture of water and methanol which was divided by phase region and used to train regression neural networks on.

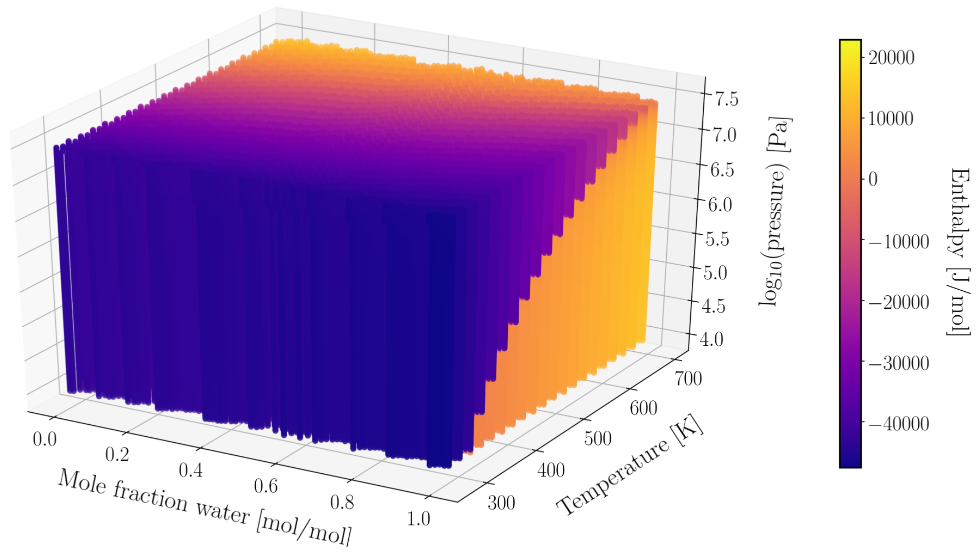


Figure 4.7: Example enthalpy data for the entire range of interest. Only a small part (∼15%) of the total data set was plotted to reduce computational burden and prevent the plot from getting difficult to interpret.

4.2.2 Data pre-processing

The section covers the details of how data was processed so that a neural network could be properly trained on it.

Data set splits

Table 4.3 shows the percentages of the total amount of data available that were divided among the training, testing, and validation data sets. The split between the data sets was done in a stratified manner such that each data set contained the same distribution between each phase, i.e. the ratios between the amounts of liquid, vapor, and two-phase region points was the same for each data set. The function used to split the data sets also simultaneously randomized the order of their instances.

In addition to splitting the entire data set into train, test and validation sets, in order to properly train each phase region regression network, the data sets were also split into sets with only liquid region instances, vapor region instances, and two-phase region instances.

Table 4.3: Percentages of the total data set included in the training, testing, and validation data sets.

Data set	Percentage [%]
Training	70
Testing	15
Validation	15

Scaling

All input and output properties apart from mole fraction and component vapor fractions were scaled to lie within the interval [0, 1] where the minimum value of the unscaled data corresponded to the value of 0 and the maximum value of the unscaled data to 1. Each property in each data set was scaled to its own maximum and minimum values. Due to their large value ranges spanning multiple decades and uneven distribution, pressure and volume values where scaled logarithmically by taking the base-10 logarithm before applying the regular normalization scaling process.

During final application of the neural networks, scaling functions were integrated with the neural networks classes in Python so that scaling was performed automatically, inputs could be given as unscaled values and outputs were also returned as such.

Formatting

Input training data was formatted in the form of an array where each column corresponded to a different input type. The first two columns always corresponded to the values of the two required thermodynamic properties (pressure, temperature, volume, entropy, enthalpy), while the last column corresponded to the value for the water mole fraction. As only two components were present at all times, only one value for mole fraction had to be specified due to fact that mole fractions have to add up to one; if one is known the other is fixed.

For the input array each row corresponded to one input combination training example, the total number of rows thus corresponded to the number of data points in the training set.

The output array was a single row (or column) array where each element was the output value corresponding to the same input row index, i.e. the n^{th} output element corresponded to the n^{th} input row.

4.2.3 Network training

In this section the settings used to construct and train the neural networks are described. All networks were created and trained using the Keras module for Python [64].

Classification networks

As mentioned, the flash problem was divided into two parts, each using a different type of neural network application. The first part of the problem is determining the phase stability of the given mixture, this is important so that the correct network can be called in during execution of the second part of the problem.

For all classification problems considered here, a feedforward multi-layer perceptron (MLP) type network was used. The target classes were given a label indicating their phase based on the value of the vapor fraction attained at each input combination: 0 for liquid, 1 for vapor, and 2 for two-phase. In order to properly train the MLP networks, the class labels were converted into a one-hot encoding [9], a type of encoding that turn a class label into a vector of the same length of the number of distinct classes in which all values are 0 except for the element at the index of the class label, which is given the value 1. For example, the liquid class with label 0 would have a one-hot encoding of [1, 0, 0], the vapor class would have an encoding of [0, 1, 0], and the two-phase class would have a label of [0, 0, 1]. The outputs of each classification network are the probabilities that a given input combination belongs to each class, based on the softmax function [9]:

$$\mathrm{sm}_i = \frac{e^{A_i}}{\sum_{j=1}^{d} e^{A_j}} \tag{4.3}$$

Here,

sm_i	is the softmax probability value of output neuron i,
A_i	is the activation of output neuron i,
A_j	is the activation of output neuron j, and
d	is the number of output neurons.

The softmax function always returns a vector of probabilities summing to one. For example, the output of the classification network might look something like [0.98, 0.002, 0.022] indicating that the network is fairly certain the input combination belongs to the first class. For all classification networks considered here, the networks final output is a label indicating which class the input combination most likely belongs too, which is inferred by determining the index of the softmax vector where the probability is highest.

The number of hidden layers was set to 1, and the number of hidden neurons restricted to 5 after it was concluded that a higher number of neurons (or hidden layers) led to significant overfitting. The sigmoid activation function was chosen for the hidden layer neurons as it could capture the curves present in the data better than the ReLu function.

All networks were trained using the categorical cross-entropy loss function as described in Equation 3.10 (Section 3.4.1 using the modified version of the *Adam* training algorithm called *Nadam* [101, 102]. The maximum number of training epochs was set to 5000. The number of training instances was 17,674,784 (70% of the total of 25,249,692 instances), where the total training data set was divided into batches of 15000 instances to be fed into the network separately.

An early-stopping monitor was used to prevent overfitting as much as possible, it monitored the validation loss and stopped the training algorithm if no decrease in the validation loss was achieved for more than 35 consecutive training epochs.

The settings described above were equal for every input combination (PTz, PSz, SVz, HVz) and are summarized in Tables 4.4, 4.5, and 4.6.

Table 4.4: Classification network topology settings.

Setting	Value
Number of input neurons	3
Number of output neurons	3
Number of hidden layers	1
Number of hidden neurons	5
Hidden activation function	Sigmoid
Output activation function	Softmax

Table 4.5: Classification network training settings.

Setting	Value
Loss function	Categorical cross-entropy
Algorithm	Nadam
Maximum training epochs	5000
Size of training set	17,674,784
Batch sizes	15000

Table 4.6: Classification network early stopping settings.

Setting	Value
Monitor	Validation loss
Minimum improvement	1×10^{-6}
Patience	35

Regression networks

The second part of the neural network approach to solving flash calculations taken in this document includes predicting values of the relevant thermodynamic properties described in Section 4.1.1 and summarized in Table 4.1.

All thermodynamic properties considered were predicted using MLP regression networks. A regression network is in principle more straight-forward than a classification network as no one-hot encoding or further inference is required. The only data processing required is scaling of the inputs and outputs to lie (roughly linearly) within the [0, 1] interval which facilitates easier learning.

As previously mentioned, each property is predicted with a different neural network. For each network the settings were exactly the same except for the amount of training data available, as much less was available for the two-phase region as for the pure liquid and vapor regions.

The only other settings different from the classification network settings were the number of hidden layer neurons (10 instead of 5 to increase accuracy of resulting network, not much overfitting occurred), the output activation function (linear as opposed to softmax), and the loss function used (mean squared error). All other settings are summarized in Tables 4.7, 4.8, and 4.9.

Table 4.7: Regression network topology settings.

Setting	Value
Number of input neurons	3
Number of output neurons	1
Number of hidden layers	1
Number of hidden neurons	10
Hidden activation function	Sigmoid
Output activation function	Linear

Table 4.8: Regression network training settings.

Setting	Value
Loss function	Mean squared error
Algorithm	Nadam
Maximum training epochs	5000
Size of vapor training set	11,149,343
Size of liquid training set	6,233,657
Size of two-phase training set	291,783
Batch sizes	500-15000

Table 4.9: Regression network early stopping settings.

Setting	Value
Monitor	Validation loss
Minimum improvement	1×10^{-6}
Patience	35

Unconventional flash types

This project included four different flash types, two of which have existing solution frameworks (the PT- and PS-flash), while the other two (the SV- and HV-flash) have, to the best of the authors knowledge, no known implemented solution framework. It was also mentioned that for a neural network, this should not be a problem. This section briefly explains why.

Neural networks can learn complex non-linear relations solely based on input-output combination examples. There is thus no need for knowledge of the underlying working principles, and as long as a data set containing enough examples is available, it is possible to accurately model a given problem.

This attribute of neural networks is used here to construct neural network based SV- and HV-flash types based on data generated using a PT-flash. Figure 4.8 shows a schematic overview of this basic principle applied to the SV-flash. First the PT-flash is used to generate entropy and volume values for a large number of pressure, temperature combinations (mole fractions are left out here as they remain constant throughout).

After the entropy and volume combinations are generated they are given as the input examples to train a neural network on, while the original pressure, temperature combinations are given as the corresponding output targets. Since a neural network only cares about input-output examples, this is a perfectly valid thing to do.

On the other hand, conventional methods are based on physical laws and principles, and their governing equations cannot as easily be reversed. One cannot use the exact same algorithm that solves a PT-flash problem to solve an SV-flash problem. The fact that a neural network can, makes them very versatile; and in theory they could be used to construct any type of flash.

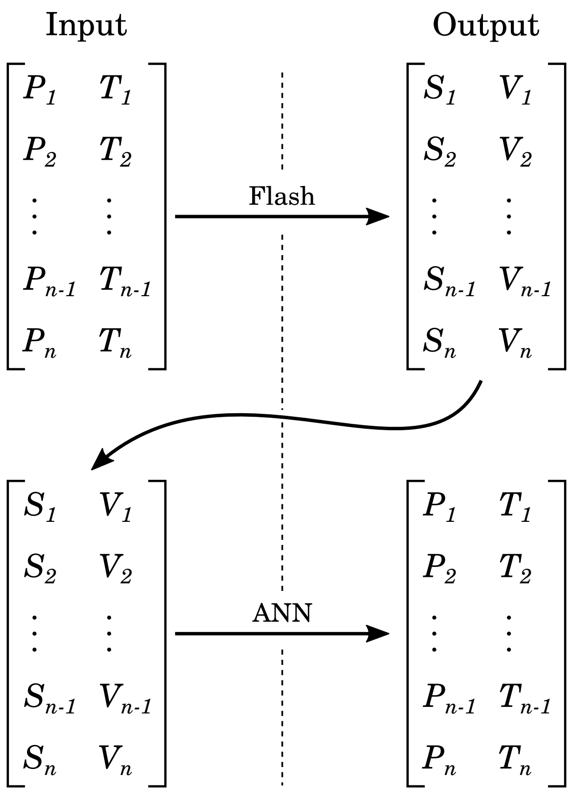

Input
Output
Flash
ANN
P_1 T_1 P_2 T_2 P_{n-1} T_{n-1} P_n T_n
S_1 V_1 S_2 V_2 S_{n-1} V_{n-1} S_n V_n

Figure 4.8: Schematic overview of the principle behind training the SV-flash. A conventional PT-flash is used to predict values of entropy (S) and volume (V) for a large number of pressure (P) and temperature (T) combinations. An artificial neural network (ANN) is subsequently trained on the entropy and volume combinations as if they were the inputs of the original problem. As the neural network does not rely on physical relations, but only on input-output combinations, it can solve the flash backwards. The method illustrated was also used for the PS- and HV-flash.

Chapter 5

Results and discussion

The goal of this project was to create an algorithm based on neural networks that could solve the classic vapor-liquid flash problem in a quicker and more robust manner compared to conventional solution methods. Once finished, the intent was for the neural network flash algorithm to be used in the dynamic simulation of a compact methanol plant. In this chapter, the current performance of the algorithm is quantified based on five different criteria: speed, accuracy, robustness, consistency, and stability. These criteria are separately discussed in the following five sections.

5.1 Speed improvement

One of the main motivations behind this research was the need for faster flash calculations, in this section the speed increase achieved through the application of the artificial neural networks will be discussed. The first part of this section briefly covers the timing methods used, after which the results will be given.

5.1.1 Speed improvement: methods

Determining execution times was done using Pythons standard time module, which contains a function that can record the current time. By subtracting the time measured directly after the desired piece of code was executed from the time measured directly before execution, the total execution time of the piece of code was determined. Execution time was determined for a total of 5 different cases:

- A conventional calculation method using COCO's TEA plugin for Python. Only a single property (vapor fraction) was predicted.
- A neural network property which requires running through only one neural network (phase classification).
- A neural network property which requires running through two separate neural networks (first phase classification, then regression).
- A neural network property which requires running through three separate neural networks (first phase classification, then two regressions).
- A single network excluding any auxiliary functions.

The last case was evaluated in order to investigate the fraction of the total execution time that is taken up by auxiliary functions required to process input and output arrays.

One of the main strengths of neural networks is that they can process a great number of input combinations all at once, whereas conventional methods can generally only process one input combination at a time, requiring the use of for-loops in order to be able to process multiple input combinations.

For this reason, it was considered of interest to investigate the scaling of execution time with the total number of input combinations to be processed. Execution times were measured for 10^1, 10^2, 10^3, 10^4, 10^5, 10^6, and 10^7 flashes to perform.

In order to mitigate the effects of randomness and noise in the time measurements, all measurements were repeated a certain number of times. The execution time of each repetition was recorded and the mean of all recorded times was calculated in order to determine the average execution time. The values shown in this text are these averages.

All time measurements were done on a MSI Leopard GP62 laptop with a 64-bit Intel®Core$^{\text{TM}}$ i7-7700HQ 2.80GHz processor, and 8.00 GB of installed ram. All speed tests were performed solely using the CPU processor using the same amount of CPU space.

5.1.2 Speed improvement: results

Table 5.1 shows the mean execution times of each case investigated for each number of flashes to execute. Figure 5.1 shows the same data in a loglog-plot, error bars were not included as the variance in execution times was in most cases to small to be noticeable in the resulting plot. In addition, Figure 5.2 shows the improvements in mean execution time of the artificial neural network predictors over the TEA predictor, calculated by dividing the execution time of TEA by that of the neural network algorithm.

Table 5.1: Average execution times of different property prediction cases for each number of flashes to execute. The different cases are: the TEA *PT*-flash algorithm [11, 31], a neural network excluding all auxiliary functions (Network only), a classification network only (Class. only), a property prediction that required executing a classification network and one prediction network (Class.+pred.) and finally a property prediction that required executing a classification network and two prediction networks (Class.+2×pred.).

	Mean execution time [s]				
No. of flashes	TEA	Network only	Class. only	Class.+pred.	Class.+2×pred.
10^1	4.28e-03	1.39e-04	7.03e-04	1.84e-03	2.64e-03
10^2	2.67e-02	1.09e-03	1.88e-03	4.82e-03	6.66e-03
10^3	3.02e-01	1.04e-02	1.16e-02	3.10e-02	4.70e-02
10^4	3.95e+00	1.07e-01	1.15e-01	3.03e-01	4.48e-01
10^5	4.20e+01	1.10e+00	1.21e+00	3.02e+00	4.69e+00
10^6	4.25e+02	1.10e+01	1.21e+01	2.95e+01	4.68e+01
10^7	1.10e+04	1.09e+02	1.22e+02	3.103+02	4.93e+02

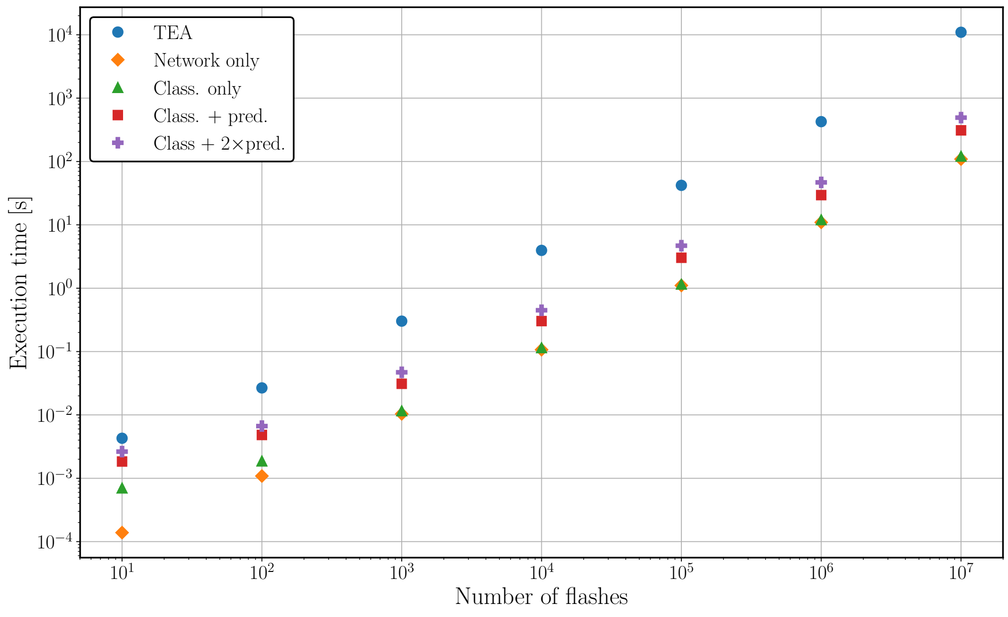


Figure 5.1: Average execution times versus the number of flashes to execute for different property prediction cases. A comparison is made between the conventional algorithm from the TEA property calculator [11, 31] (blue circles), a neural network excluding auxiliary functions (orange diamonds), a classification neural network only (green triangles), a property which required execution of the classification network and one regression network (red squares), and a property which required execution of the classification network and two regression networks (purple plusses).

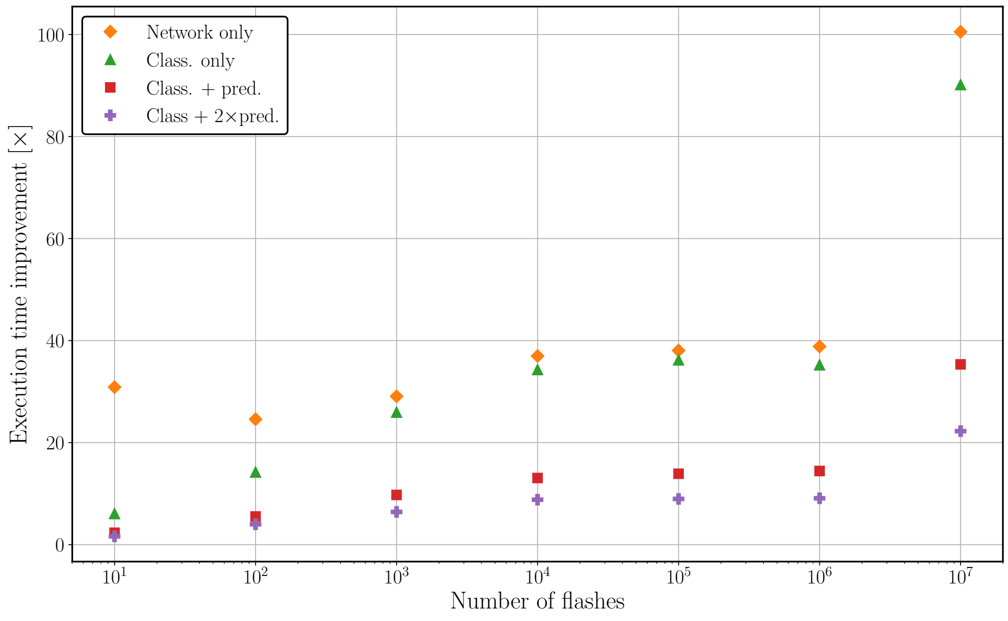


Figure 5.2: Average execution time improvement over the conventional algorithm from TEA [11, 31] versus the number of flashes to execute, calculated by dividing the average execution time of the TEA property calculator by the average execution time of the neural network. A comparison is made between a neural network excluding auxiliary functions (orange diamonds), a classification neural network only (green triangles), a property which requires execution of the classification network and one regression network (red squares), and a property which requires execution of the classification network and two regression networks (purple plusses).

As can be seen from Figure 5.1, execution time increases roughly linearly with the total number of samples to generate on a log-log-scale. The classification, classification and regression, and classification and double regression cases deviate slightly from this linear trend at lower sample numbers, while the TEA execution time deviates from linear at higher samples numbers (especially for the number 10^7). Table 5.2 shows the coefficients of the power-law fits $y = A \times x^m$ to the data in Figure 5.1 as well as the R^2-scores of these fits, it shows that most cases indeed scale roughly linearly.

Table 5.2: Coefficients of the fit $y = A{\times}x^m$ made on the data shown in Figure 5.1 and the R^2-scores of the fits for the different property prediction cases. The different cases are: the TEA *PT*-flash algorithm [31], a neural network excluding all auxiliary functions, a classification network only, a property prediction that required executing a classification network and one prediction network, and finally a property prediction that required executing a classification network and two prediction networks.

Prediction case	A	m	R^2-score
TEA	2.3942e-04	1.063	0.816
Network only	1.2272e-05	0.990	0.998
Class. only	3.9160e-05	0.911	0.886
Class.+pred.	1.0400e-04	0.901	0.881
Class.+2×pred.	1.4342e-04	0.904	0.887

Figure 5.2 shows a less clear trend between speed improvement and number of flashes, but, except for the network only case, all other ANN cases show an increasing improvement with the number of samples. While the improvements seem to saturate at flash numbers at and above 10^4, all show a sudden increase for the 10^7 flash case due to a large increase in the TEA execution time at 10^7 flashes. Why the TEA execution time at that number of flashes increases so much in comparison to the other numbers is mostly unknown. The value is an average of multiple executions, so it cannot be a data outlier, and it used the same amount of CPU space as all other values. It might result from the way in which Python handles filling large data arrays, which could slow down severely for large arrays, although the author does not have enough knowledge of this aspect of Python to conclude for certain whether this might be the case.

The results also show that execution times significantly increase when going from one network to execute to two networks to execute, but going to three networks shows a much lower effect on execution time. The reason for the jump in execution time duration from classification only to classification followed by prediction is, naturally, due to the fact that the number of networks to execute doubles. Furthermore, certain auxiliary functions that weren't necessary for the classification only case have to be executed for the classification and prediction case, adding additional time. For the classification and double prediction case, the auxiliary function were only performed for the first prediction case, so the increase in execution time is less drastic and only due to the extra network that has to be executed.

Since the majority of properties included in the flash algorithm require two networks to be executed, first phase classification, then property value prediction, overall speed improvements lie around the 13-15 times mark at higher numbers of flashes to execute (10^4 and higher). Phase classification itself shows an improvement of around 34-35 times. In general, as was expected, the neural network approach shows much fewer benefits over conventional methods at lower numbers of flashes to execute (10^3 and fewer).
Finally, Figure 5.1 and the data in Table 5.1 also show that the execution times not

taking into account auxiliary functions are much faster at lower flash numbers, but are comparable to the single network times at numbers of 10^3 and higher. From the raw data in Table 5.1 is has been calculated that at low flash numbers, auxiliary functions can take up to 80% of the total execution time, while at higher numbers they only contribute 5-10% of the total execution time.

5.2 Accuracy quantification

In this section the results and discussion of the accuracy scores of the different neural networks are covered. Firstly, the way in which accuracy was determined and which accuracy metrics were used will be briefly discussed, after which phase classification and property prediction accuracy scores will be discussed in separate sections.

5.2.1 Accuracy quantification: methods

In order to determine the accuracy of the trained networks, three additional data sets containing data points that the networks had never encountered before were generated; one for each phase region. The new data was generated so that any bias that was introduced by the training data generation method used (and was thus present in the original test data set) would come to light. The three test data sets all contained 100,000 randomly generated PTz-combinations and their corresponding values for phase, volume (V), enthalpy (H), entropy (S), internal energy (U), Gibbs free energy (G), and Helmholtz free energy (A).

Property calculations were done using the TEA plugin for Python [11]. The pressure, temperature and mole fraction combinations were determined by randomly generating a value for each property within the boundaries as previously specified in Table 4.2 in Section 4.2.1.

Accuracy was determined by giving relevant property values in the test data set as inputs to each neural network flash and comparing the predicted values of each property to the target property values also in the test data set.

The phase classification networks were scored based on a binary accuracy (BA) comparing the number of correct predictions against the total number of predictions made:

$$\mathrm{BA} = \frac{n_{\mathrm{cor}}}{n} \times 100 \tag{5.1}$$

Here,

BA	is the binary accuracy score in [%],
n_{cor}	is the number of correct predictions, and
n	is the total number of predictions made.

For the regression networks, the main accuracy metrics used were the mean absolute error (MAE), and the coefficient of determination, also known as the R^2 score, these metrics are defined as:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |t_i - o_i| \tag{5.2}$$

$$\mathrm{R}^2 = 1 - \frac{\sum_{i=1}^{n-1}(t_i - o_i)^2}{\sum_{i=1}^{n-1}(t_i - \mu_{\mathrm{test}})^2} \tag{5.3}$$

Here,

MAE	is the mean absolute error,
R^2	is the coefficient of deviation,
t_i	is the target output value of instance i,
o_i	is the predicted output value of instance i, and
μ_{test}	is the mean value of the target data set.

For each network the MAE was first determined for the regular test data, such that the MAE metric and data would have the same unit. While the MAE gives a good idea of the size of the absolute deviation, is hard to interpret without knowing the relative magnitude of the raw data. For example, an MAE of 1 is quite accurate when predicting values in the range of [0, 100], however, if data is predicted which lies in the range of [0, 5], an MAE of 1 is much less accurate.

For this reason, the results below also give the mean value (μ_{test}) and standard deviation (σ_{test}) of the test data, to indicate the relative size and spread within it. In addition, the MAE is also given as a percentage of the standard deviation, to give an idea on how the MAE compares to the spread within the data set.

Lastly, the MAE was also determined for the same data as before, but scaled to be within the range [0, 1], which made it possible to compare the relative accuracy scores between networks trained on different properties having different units.

In many articles and reports, accuracy is quantified using the mean absolute percentage error (MAPE) in order to give an easily interpretable measure of accuracy. However, in this thesis the decision was made to not use the MAPE as an accuracy metric as it has a number of disadvantages: (i) the MAPE is biased towards underpredictions (equal valued predictions under the target have a lower MAPE value than above the target); (ii) when the target value is close to zero, the resulting absolute percentage error can be disproportionately high and severely skew the total MAPE; and (iii) very high prediction errors can also disproportionately skew the MAPE [103]. In addition, the MAPE is only valid on a ratio scale, and since values of enthalpy, entropy, Gibbs free energy, Helmholtz free energy and internal energy are all measured relative to some reference state, they don't have an absolute zero point. As a result, the MAPE should not be used to score these properties.

For a similar reason, the MAE as a percentage of the standard deviation must also be taken with a grain of salt, although it gives a less skewed indication of accuracy than the MAPE would. The reason the MAE as a percentage of σ_{test} was still included, was that percentage scores generally give a more easily interpretable score that can be used to compare multiple different situations with each other. Therefore, the MAE percentage is best interpreted alongside the values of the MAE, scaled MAE, and R^2 scores to see whether they show similar trends. For instance, if the MAE percentage is very low, but the R^2 and scaled MAE scores are poor, there's probably something going on that skews the MAE percentage. The results in the next sections show that for most cases, trends do match.

5.2.2 Accuracy quantification: phase classification results

Each phase classification artificial neural network was given 4 binary accuracy scores, one for each phase region, and one for the total combined data sets. For all incorrect predictions, the distribution between the remaining two phases was determined in order to get an idea along which phase boundaries most errors occur for each phase.

Pressure, temperature flash classification accuracy

Accuracy scores for the constant pressure, constant temperature flash are given in Table 5.3. As can be seen, the PT classifier has high accuracy scores on the vapor and liquid phases, but scores significantly lower on the two-phase region test set, which has a considerable impact on the value of the overall score.

The reason for the comparably low scores on the two-phase region data set are most likely due to the lower amount of training data that was generated for the two-phase region. As mentioned in Section 4.2, data was generated on an evenly/logarithmically spaced grid between desired boundaries. This approach, combined with the fact that the two-phase region takes up only a small part of the entire area of interest (see Figure 4.5) resulted in much less data being available for points within the two-phase region when compared to the amount of data points in the liquid and vapor only regions. And since the accuracy of a neural network is largely determined by the amount of data that is available, less data in the two-phase region likely led to a less accurate predictions in this region.

Table 5.4 shows a so-called confusion matrix of the PT-flash classifier. In machine learning, a confusion matrix is used to show whether (and to which extend) a classifier struggles to distinguish between classes. For the confusion matrix in Table 5.4, labels on the left column indicate the correct class, while the labels on the upper row indicate the class that was predicted by the classifier. For example, off all examples belonging to the liquid class, 99.8% were correctly classified as liquid, while 0.194% were incorrectly classified as belonging to the vapor class, and only 0.006% as belonging to the two-phase class.

Table 5.4 further shows that almost all errors made in the liquid region are misclassified as the belonging to the vapor region, while for the vapor region most misclassifications are classified as belonging to the liquid region. Errors in the two-phase region are more evenly distributed, with some bias towards the vapor phase. From this it can be concluded that the classifier more often mistakes the vapor and liquid regions with each other than with the two-phase region, while the two-phase region is confused more often with the vapor region than the liquid region, although the difference is less pronounced.

While the vapor and liquid regions show a significant bias towards each other when it comes to incorrect classifications, the total number of misclassifications is so small that it isn't considered worth the trouble to try and mitigate this bias. For the two-phase region accuracy should be improved in general, but since the bias is less pronounced, improvements can't be targeted much towards solving a certain bias.

Table 5.3: Pressure, temperature flash phase classification accuracy per phase, expressed as the number of correct classifications as a percentage of the total number predictions made.

Phase region	Accuracy [%]
Liquid	99.8
Vapor	99.7
Two-phase	91.0
Overall	96.8

Table 5.4: Pressure, temperature flash phase classification confusion matrix. The labels on the left column indicate the correct classes, while the labels on the upper row indicate the classes that were predicted. All numbers are given as percentages of the total number of predictions made.

	Liquid	Vapor	Two-phase
Liquid	**99.8**	0.194	0.006
Vapor	0.276	**99.7**	0.024
Two-phase	3.69	5.31	**91.0**

Pressure, entropy flash classification accuracy

Tables 5.5 and 5.6 show the constant pressure, constant entropy flash classifier accuracy scores and confusion matrix respectively. The accuracy score table shows similar results as the PT-flash tables showed, higher accuracy for the liquid and vapor regions, and lower accuracy for the two-phase region. The confusion matrix also shows roughly similar distributions for the incorrect predictions as the PT-flash showed. The lower accuracy of two-phase classification is again most likely due to the lower amount of two-phase region training examples.

Table 5.5: Pressure, entropy flash phase classification accuracy per phase, expressed as the number of correct classifications as a percentage of the total number predictions made.

Phase	Accuracy [%]
Liquid	99.7
Vapor	99.7
Two-phase	93.4
Overall	97.6

Table 5.6: Pressure, entropy flash phase classification confusion matrix. The labels on the left column indicate the correct classes, while the labels on the upper row indicate the classes that were predicted. All numbers are given as percentages of the total number of predictions made.

	Liquid	Vapor	Two-phase
Liquid	**99.7**	0.291	0.009
Vapor	0.294	**99.7**	0.006
Two-phase	3.1	3.5	**93.4**

Entropy, volume flash classification accuracy

Tables 5.7 and 5.8 show the accuracy scores and confusion matrix of the constant entropy, constant volume flash classifier. Again, similar accuracy results are seen as in the previous accuracy score tables, although both the liquid and vapor region predictions are much more accurate.

This time, the confusion matrix is more different than the ones previously shown. As can be seen, in this case the liquid region is never confused with the two-phase region, and only in a few instances with the vapor region. On the other hand, the vapor region is now more often confused with the two-phase region, although the total vapor classification accuracy is much higher. Lastly, the two-phase region is in this case more often mixed up with the liquid region than the vapor region, although again, the biases are not very pronounced.

Table 5.7: Entropy, volume flash phase classification accuracy per phase, expressed as the number of correct classifications as a percentage of the total number predictions made.

Phase	Accuracy [%]
Liquid	99.9
Vapor	99.99
Two-phase	91.0
Overall	97.0

Table 5.8: Entropy, volume flash phase classification confusion matrix. The labels on the left column indicate the correct classes, while the labels on the upper row indicate the classes that were predicted. All numbers are given as percentages of the total number of predictions made.

	Liquid	Vapor	Two-phase
Liquid	**99.9**	0.01	0.00
Vapor	0.0014	**99.99**	0.0086
Two-phase	5.94	3.06	**91.0**

Enthalpy, volume phase classification accuracy

Tables 5.9 and 5.10 show the accuracy scores and confusion matrix of the constant enthalpy, constant volume flash classifier. Accuracy scores are comparable to the SV-flash case, but again much higher for the liquid and vapor regions. Most notable is the fact that the liquid region is always predicted correctly, and the vapor region in only very few instances. On the other hand, two-phase region predictions are still on the same level as previous accuracy scores showed.

In this case the confusion matrix does not provide much additional insight apart from the fact that the two-phase errors are fairly equally distributed between the liquid and vapor regions.

Table 5.9: HV-flash phase classification accuracy per phase, expressed as the number of correct classifications as a percentage of the total number predictions made.

Phase	Accuracy [%]
Liquid	100.0
Vapor	99.997
Two-phase	90.1
Overall	96.7

Table 5.10: HV-flash phase classification confusion matrix. The labels on the left column indicate the correct classes, while the labels on the upper row indicate the classes that were predicted. All numbers are given as percentages of the total number of predictions made.

	Liquid	Vapor	Two-phase
Liquid	**100.0**	0.00	0.00
Vapor	0.003	**99.997**	0.00
Two-phase	5.45	4.45	**90.1**

5.2.3 Accuracy quantification: property value prediction results

In this subsection the results from the thermodynamic property value regression neural networks will be discussed. Each flash type will be covered separately.

Pressure, temperature flash accuracy scores

Tables 5.11, 5.12, and 5.13 show the test data mean value (μ_{test}) and standard deviation (σ_{test}), mean absolute error of the neural network predictions (MAE), the mean absolute error as a percentage of the standard deviation (MAE%), the scaled mean absolute error (MAE$_{\text{sc}}$), and finally the coefficient of deviation (R^2) for the liquid, vapor, and two-phase region regression neural networks of the PT-flash. The scores of the two-phase compositions x_i and y_i were based on the accuracy of the water mole fraction x_{w} and y_{w}. Appendix F shows that for the mean absolute error there should be no difference whether accuracy is calculated for the water or methanol fraction, however, percentage and R^2 scores will differ, but are similar in magnitude. It was therefore decided to only show the water mole fraction accuracy scores.

Table 5.11: Pressure, temperature flash: **liquid** region accuracy scores. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). Additionally, the mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
V [m^3/mol]	4.19e-05	2.25e-05	7.67e-07	3.41e+00	7.17e-03	9.790e-01
H [J/mol]	-2.79e+04	1.23e+04	1.97e+02	1.61e+00	3.28e-03	9.987e-01
S [J/mol/K]	-8.17e+01	2.69e+01	4.10e-01	1.52e+00	3.43e-03	9.991e-01
G [J/mol]	4.77e+03	7.96e+03	9.85e+01	1.24e+00	2.83e-03	9.997e-01
A [J/mol]	4.01e+03	7.59e+03	1.05e+02	1.38e+00	3.12e-03	9.996e-01
U [J/mol]	-2.87e+04	1.18e+04	1.29e+02	1.09e+00	2.29e-03	9.993e-01

Table 5.12: Pressure, temperature flash: **vapor** region accuracy scores. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). Additionally, the mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
V [m^3/mol]	2.50e-03	1.37e-02	2.08e-05	1.52e-01	3.91e-05	9.999e-01
H [J/mol]	9.72e+03	4.33e+03	2.76e+02	6.38e+00	4.19e-03	9.866e-01
S [J/mol/K]	-4.11e+00	1.28e+01	4.25e-01	3.32e+00	2.30e-03	9.960e-01
G [J/mol]	1.23e+04	5.44e+03	1.51e+02	2.77e+00	3.17e-03	9.986e-01
A [J/mol]	8.29e+03	5.78e+03	1.14e+02	1.97e+00	2.26e-03	9.991e-01
U [J/mol]	5.67e+03	3.61e+03	2.47e+02	6.83e+00	4.03e-03	9.861e-01

Table 5.13: Pressure, temperature flash: **two-phase** phase region accuracy scores. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE_{sc}), and the coefficient of deviation (R^2). Additionally, the mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE_{sc}	R^2
V [m^3/mol]	1.24e-03	5.39e-03	2.11e-04	3.92e+00	1.07e-03	9.664e-01
H [J/mol]	-9.10e+03	8.85e+03	1.26e+03	1.42e+01	2.36e-02	9.431e-01
S [J/mol/K]	-3.81e+01	1.81e+01	2.65e+00	1.47e+01	1.96e-02	9.479e-01
G [J/mol]	9.89e+03	4.34e+03	5.54e+01	1.28e+00	1.94e-03	9.995e-01
A [J/mol]	8.26e+03	4.43e+03	1.76e+02	3.96e+00	6.03e-03	9.953e-01
U [J/mol]	-1.07e+04	8.11e+03	9.89e+02	1.22e+01	1.98e-02	9.578e-01
β_{w} [mol/mol]	4.58e-01	2.88e-01	5.25e-02	1.82e+01	5.25e-02	8.821e-01
β_{m} [mol/mol]	5.94e-01	2.86e-01	6.68e-02	2.34e+01	6.68e-02	8.707e-01
x_{w} [mol/mol]	7.19e-01	2.09e-01	7.14e-02	3.42e+01	5.79e-02	9.823e-02
y_{w} [mol/mol]	5.77e-01	2.16e-01	6.25e-02	2.89e+01	5.95e-02	4.124e-01

As can be seen from the tables, accuracy scores for both the pure liquid and pure vapor regions are relatively good, with comparable scaled MAE and R^2 scores, and regular MAE scores whose magnitudes are only a few percentage points of the average spread of the magnitude of the test data. Only the liquid volume, and vapor enthalpy and internal energy predictions are somewhat less accurate.

The lower score on liquid volume is likely due to the phase classifier classifying a point that should be in the liquid region as being in the vapor region. As the value of volume shows large differences depending on the phase region, incorrectly classifying a liquid point for a vapor point will lead the algorithm to use the vapor region neural network, which will generally predict a much larger value for volume than the liquid region neural network would. As a result, the incorrect classifications lead to higher errors. This can also be seen from the liquid volume correlation plot shown in Figure 5.3. This figure plots the correct target value of volume versus the actual value predicted by the neural network, and clearly shows that whenever the classification network incorrectly predicts the phase (red crosses), the resulting volume value prediction is quite far removed from the target value. Compare this to the correlation plot for the vapor region volume shown in Figure 5.4, which contains very few misclassified instances, and the effect incorrect classifications can have on property value predictions becomes evident.

For additional reference, Table 5.14 shows the percentage of the total MAE that resulted from incorrect classifications of the properties considered here for each phase region. The percentage was determined using the following equation:

$$\text{ECM} = \frac{\text{MAE}_{\text{tot}} - \text{MAE}_{\text{cor}}}{\text{MAE}_{\text{tot}}} \times 100 \tag{5.4}$$

Here,

ECM	is the error contribution of misclassifications in [%],
MAE_{tot}	is the MAE of the total data set including misclassifications, and
MAE_{cor}	is the MAE of the data set that only included correctly classified instances.

The exact accuracy scores excluding incorrect classifications can be found in Appendix H. As can be seen from Table 5.14, if all misclassifications of the liquid volume are excluded, the total MAE would be around 10% lower. Additionally, the table also shows that incorrect classifications in the two-phase region contribute a lot to the total MAE of many properties, especially for the values of x_w and y_w.

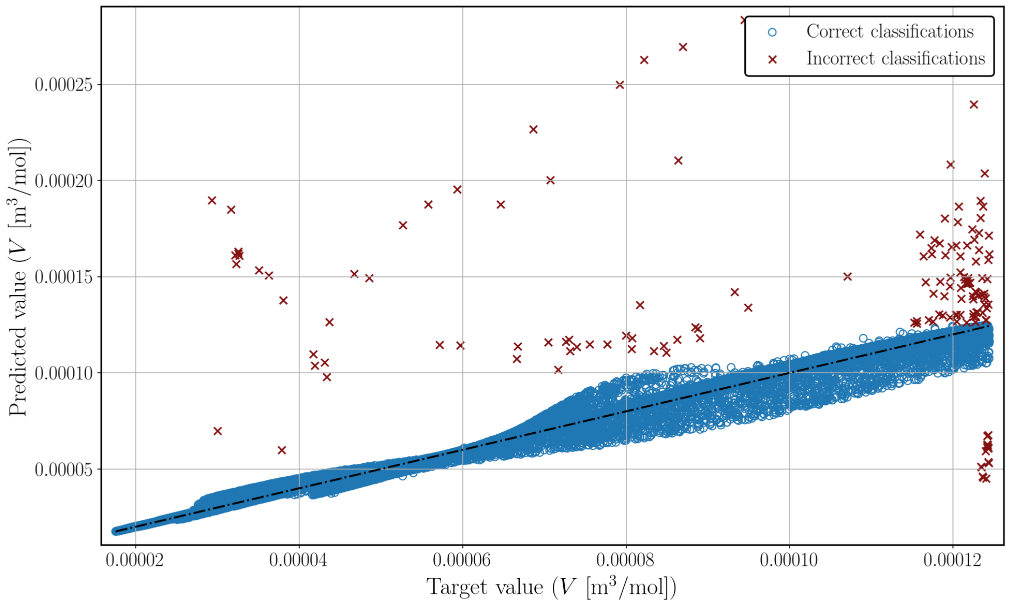


Figure 5.3: Pressure-temperature flash: liquid volume prediction correlation plot. The graph shows the target volume value in the test data set versus the volume value predicted by the neural network. The black dashed line indicates the line of perfect correlation (prediction = target). As can be seen, if only correct phase classifications are included, correlation is decent, but incorrect classification lead to much higher errors, and a somewhat low coefficient of deviation (R^2 = 0.970).

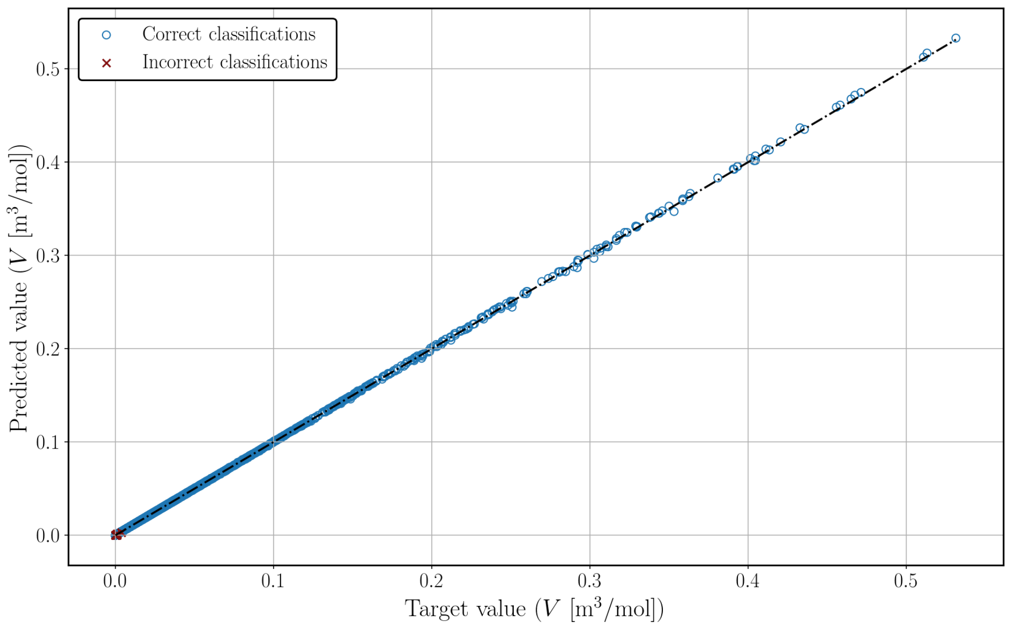


Figure 5.4: Pressure-temperature flash: vapor volume prediction correlation plot. The graph shows the target volume value in the test data set versus the volume value predicted by the neural network. The black dashed line indicates the line of perfect correlation (prediction = target). As can be seen, there are few incorrect classification, and those that are present, do not affect the overall correlation much, which is very high (R^2 = 0.9999).

Table 5.14: Pressure-temperature flash: percentage of the mean absolute error that resulted from instances whose phase was incorrectly classified (ECM). Calculated as per Equation 5.4.

	ECM [%]		
Property	Liquid	Vapor	Two-phase
V	10.31	0.56	7.49
H	1.39	2.92	11.89
S	1.20	3.30	8.61
G	0.64	0.28	25.80
A	1.51	1.94	19.40
U	1.97	2.03	13.33
β_{w}	-	-	21.01
β_{m}	-	-	12.75
x_{w}	-	-	49.44
y_{w}	-	-	28.58

The results for the two-phase region given in Table 5.13 show comparably higher error scores than the vapor and liquid regions, with only volume, and Gibbs and Helmholtz free energy being predicted with relatively high accuracy. The reason for the higher error scores is most likely due to multiple reasons: first, as mentioned previously, less training data was available for the two-phase region, secondly, two-phase region phase classification is less accurate than for the liquid and vapor regions, which in turn leads to higher errors in two-phase region property prediction, and finally, the PT two-phase region shows large changes in property values, which make it more difficult for an artificial neural network to correctly predict output values.

The last point is illustrated with Figure 5.5 and Figure 5.6, which show the data sets generated for enthalpy and Gibbs free energy respectively, for a 50-50 mol% mixture of water and methanol. As can be seen, the first of the two plots shows a very steep change in enthalpy across the two-phase region (shown in green), while the second plot shows a relatively smooth change in Gibbs free energy within the two-phase region. The steep gradient present in the first plot results in a large difference in distribution between inputs, which are very close together in value, and outputs, which are very far apart. This makes it difficult for a neural network to properly find the relations between inputs and outputs. Unfortunately, simply linearly normalizing the data does not solve the problem, as it doesn't change the distributions of inputs and outputs.

As Helmholtz free energy shows a similar data plot as Gibbs free energy, it explains why it's also predicted with higher accuracy. Along with all other properties, volume shows a curve similar to that of enthalpy shown in Figure 5.5, but shows higher accuracy than most of the other properties. The reason for this cannot be explained fully by the author, but could be due to scaling of the training data. As opposed to the other properties, volume was scaled logarithmically, this could have resulted in the data being easier to train a neural network on. Due to the fact that this effect was unknown to the author, it was not done for the other properties, which would explain the differences in error scores.

The next section will explain the large difference in accuracy between the component vapor fractions (β_i) and the two-phase region liquid and vapor compositions (x_{w} and y_{w}).

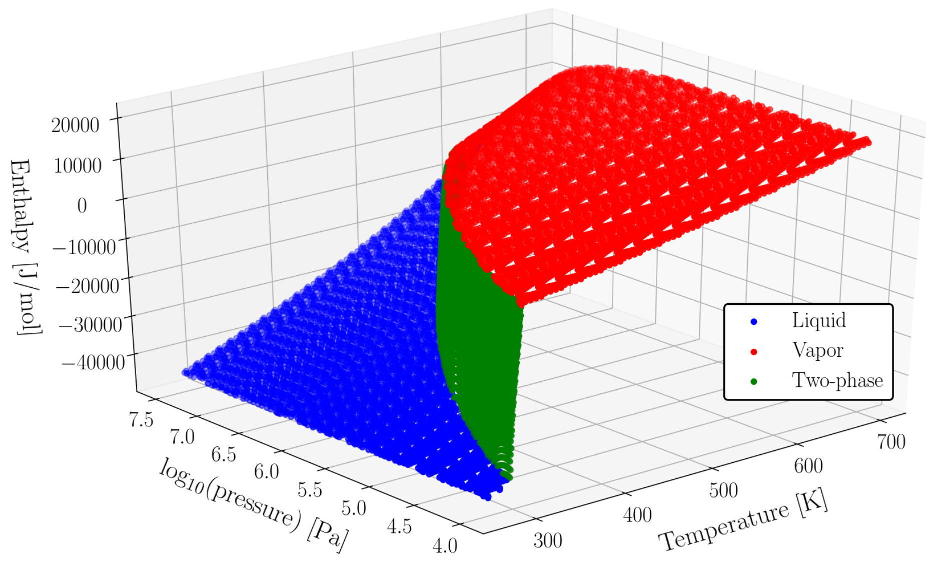


Figure 5.5: Three-dimensional scatter-plot of a part of the data set (single composition) that was used to train the *PT*-flash enthalpy regression neural networks. Data is shown for the liquid region (blue), the vapor region (red) and the two-phase region (green). The visual curve-like structures that seem present in the data sets result from the fact that only a certain percentage of all data points are plotted (in order to reduce computational burden on the plotting software).

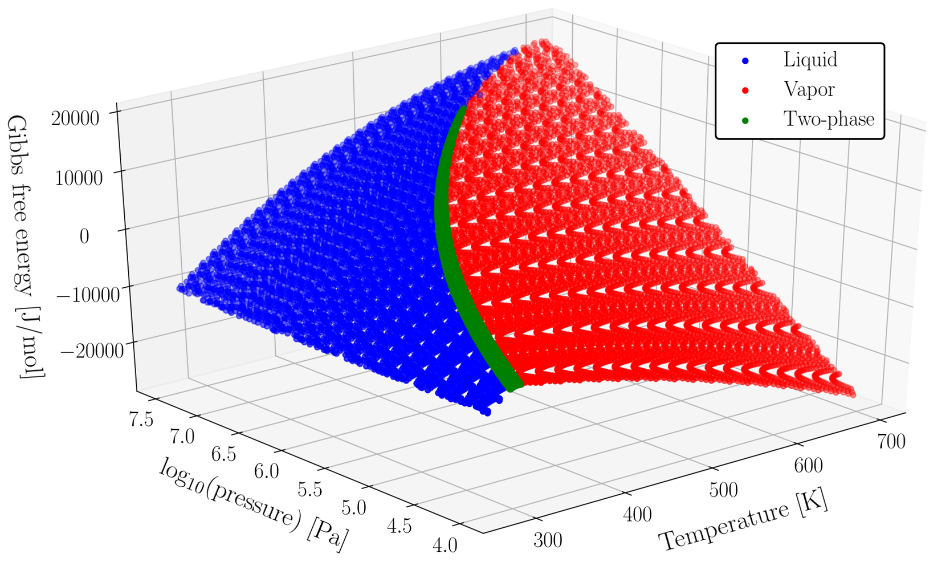


Figure 5.6: Three-dimensional scatter-plot of a part of the data set (single composition) that was used to train the *PT*-flash Gibbs free energy regression neural networks. Data is shown for the liquid region (blue), the vapor region (red) and the two-phase region (green). The visual curve-like structures that seem present in the data sets result from the fact that only a certain percentage of all data points are plotted (in order to reduce computational burden on the plotting software).

Appendix I contains an additional selection of correlation plots. A number of the plots, most notably Figure I.1, I.3, and I.11 show that predictive accuracy becomes lower as the the magnitude of the property goes towards a maximum or minimum (depending on the property). A likely reason for this is that less data was generated for high pressures as compared to low pressure (for reasons given in Section 4.2). As a result, data for extreme values of other properties are also less prevalent, which can explain why predictive accuracy becomes lower towards the extreme values of certain properties. Similar trends occur for the other flash types.

Furthermore, for many properties (most notably enthalpy, Gibbs free energy, Helmholtz free energy, and internal energy) the correlation plots in Appendix I also show that at extreme values, incorrect classifications often don't lead to very high differences in property value predictions. This is due to the fact that these extremes occur near or above the critical point of the water-methanol mixture, which is a region where the boundaries between what is liquid and what is vapor start to disappear, and similarly property values differ little from one region to the other.

Pressure, entropy flash accuracy scores

Tables 5.15, 5.16 and 5.17 show the test data mean value (μ_{test}) and standard deviation (σ_{test}), and the mean absolute error (MAE), mean absolute error as a percentage of the standard deviation (MAE%), the scaled mean absolute error (MAE$_{\text{sc}}$), and finally the coefficient of deviation (R^2) for the liquid, vapor, and two-phase region regression neural networks of the PS-flash.

Table 5.15: Pressure, entropy flash: **liquid** region accuracy scores. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). Additionally, the mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
T [K]	4.31e+02	9.82e+01	1.15e+00	1.17e+00	2.76e-03	9.994e-01
V [m^3/mol]	4.19e-05	2.25e-05	6.14e-07	2.73e+00	5.74e-03	9.797e-01
H [J/mol]	-2.79e+04	1.23e+04	1.26e+02	1.03e+00	2.10e-03	9.998e-01
G [J/mol]	4.77e+03	7.96e+03	1.27e+02	1.60e+00	3.67e-03	9.995e-01
A [J/mol]	4.01e+03	7.59e+03	1.55e+02	2.04e+00	4.60e-03	9.992e-01
U [J/mol]	-2.87e+04	1.18e+04	9.03e+01	7.66e-01	1.60e-03	9.998e-01

Table 5.16: Pressure, entropy flash: **vapor** region accuracy scores. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). Additionally, the mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
T [K]	6.19e+02	6.23e+01	2.95e+00	4.73e+00	7.13e-03	9.944e-01
V [m^3/mol]	2.50e-03	1.37e-02	4.39e-05	3.21e-01	8.26e-05	9.993e-01
H [J/mol]	9.72e+03	4.33e+03	1.30e+02	3.01e+00	1.98e-03	9.983e-01
G [J/mol]	1.23e+04	5.44e+03	1.86e+02	3.42e+00	3.92e-03	9.976e-01
A [J/mol]	8.29e+03	5.78e+03	1.67e+02	2.90e+00	3.32e-03	9.980e-01
U [J/mol]	5.67e+03	3.61e+03	2.26e+02	6.25e+00	3.69e-03	9.933e-01

Table 5.17: Pressure, entropy flash: **two-phase** region accuracy scores. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE_{sc}), and the coefficient of deviation (R^2). Additionally, the mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE_{sc}	R^2
T [K]	5.03e+02	5.92e+01	1.23e+00	2.08e+00	3.10e-03	9.972e-01
V [m^3/mol]	1.24e-03	5.39e-03	2.57e-05	4.77e-01	1.30e-04	9.995e-01
H [J/mol]	-9.10e+03	8.85e+03	1.17e+02	1.32e+00	2.19e-03	9.996e-01
G [J/mol]	9.89e+03	4.34e+03	8.77e+01	2.02e+00	3.07e-03	9.987e-01
A [J/mol]	8.26e+03	4.43e+03	1.13e+02	2.56e+00	3.90e-03	9.984e-01
U [J/mol]	-1.07e+04	8.11e+03	8.83e+01	1.09e+00	1.77e-03	9.997e-01
β_{w} [mol/mol]	4.58e-01	2.88e-01	1.01e-02	3.49e+00	1.01e-02	9.931e-01
β_{m} [mol/mol]	5.94e-01	2.86e-01	1.54e-02	5.37e+00	1.54e-02	9.887e-01
x_{w} [mol/mol]	7.19e-01	2.09e-01	3.61e-02	1.73e+01	2.67e-02	4.644e-01
y_{w} [mol/mol]	5.77e-01	2.16e-01	3.43e-02	1.58e+01	3.10e-02	6.212e-01

As can be seen, almost all networks show good accuracy scores. Liquid volume again shows a somewhat higher scaled MAE and lower R^2 score. As opposed to the PT-flash, the PS-flash shows overall quite low error scores in the two-phase region, apart from the vapor and liquid phase compositions. This is very likely due to the fact that the PS two-phase region does not contain the steep property value gradients that the PT region showed (see Figure 5.7).

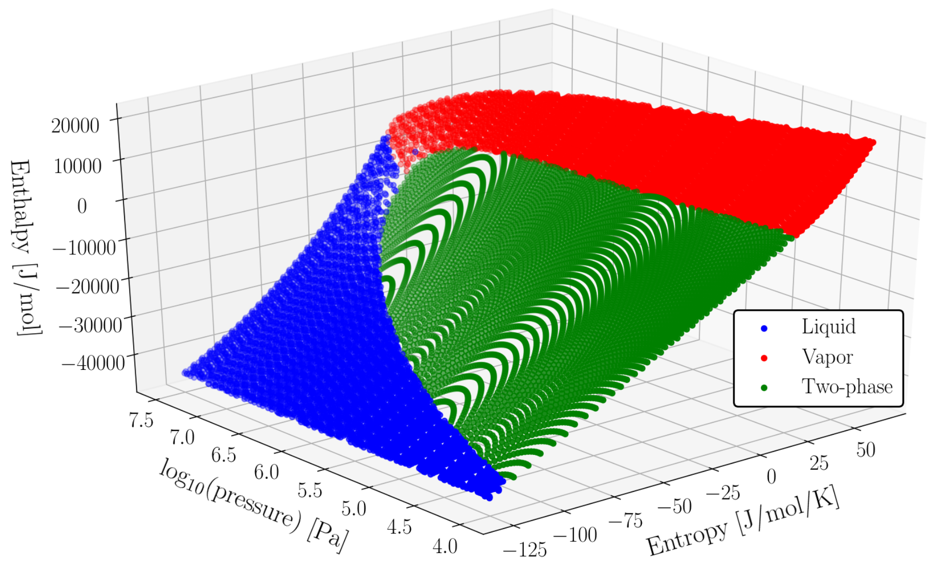


Figure 5.7: Three-dimensional scatter-plot of a part of the data set (single composition) that was used to train the PS-flash enthalpy regression neural networks. Data is shown for the liquid region (blue), the vapor region (red) and the two-phase region (green). The visual curve-like structures that seem present in the data sets result from the fact that only a certain percentage of all data points are plotted (in order to reduce computational burden on the plotting software).

As can be seen from Table 5.17, predictions of the two-phase compositions x_{w} and y_{w} are much less accurate compared to all other properties. The reason for this is best explained using correlation plots similar to the plots shown in Figures 5.3 and 5.4.

Figure 5.8 shows the PS-flash correlation plot for the two-phase vapor composition mole fraction of water y_w, while Figure 5.9 shows the correlation plot of one of the component vapor fractions β_w from which y_w was in part derived (using Equations 4.2 from Section 4.1.1). The value for β_w was what was actually predicted using a neural network. As can be seen, the correlation of the component vapor fraction is much better than that of the vapor phase composition. The conclusion can be drawn that in the conversion from β_w (and β_m) to y_w (and x_w) additional errors are introduced, most likely because two separate and independent networks are used to derive properties that are dependent on each other. Thus, the errors from one network are added to the errors of the other network to give even larger errors in their derived properties. This is also shown mathematically in Appendix F).

The other flash types show the same trend of the actual component vapor fraction values being predicted fairly accurately, while the derived values of phase composition show much higher errors.

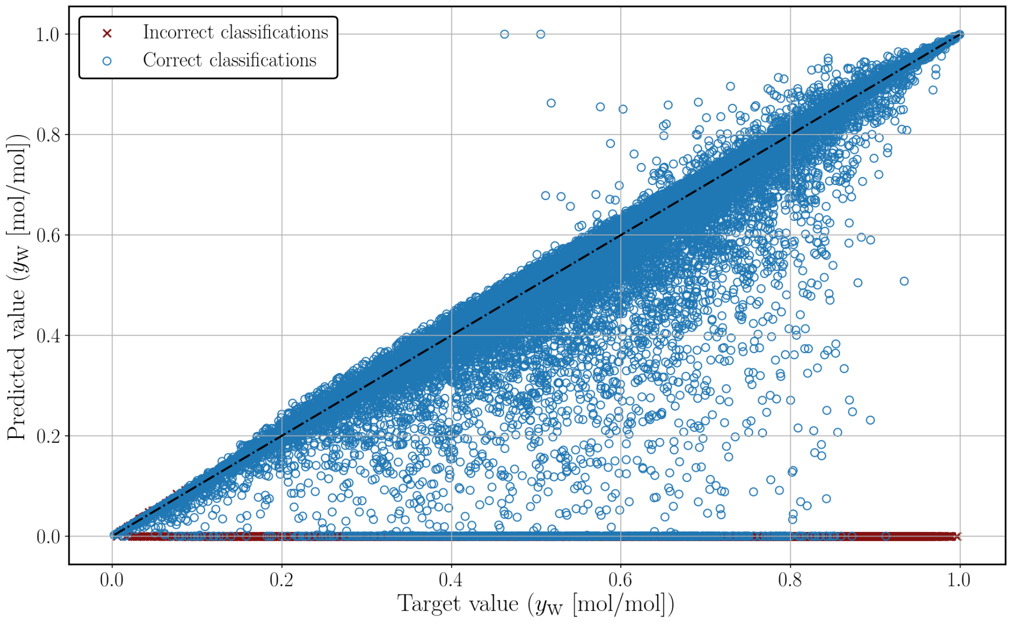


Figure 5.8: Pressure-entropy flash: two-phase vapor phase mole fraction of water (x_w) prediction correlation plot. The graph shows the target value of the vapor phase mole fraction of water from the test data set versus the value of the vapor phase mole fraction of water predicted by the neural network. The black dashed line indicates the line of perfect correlation (prediction = target). The data shown has a coefficient of deviation of $R^2 = 0.4124$.

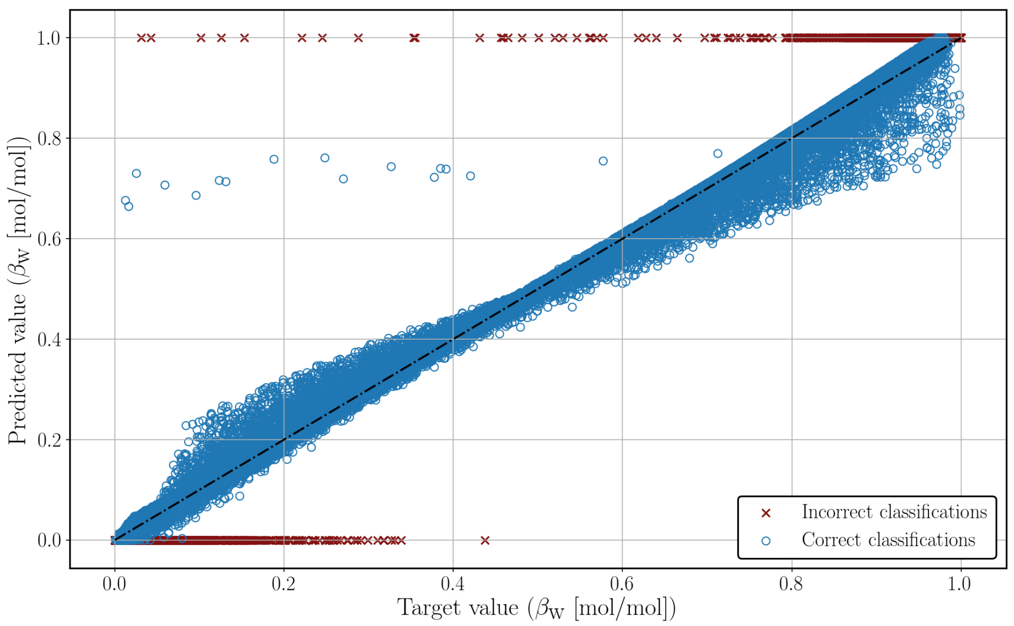


Figure 5.9: Pressure-entropy flash: two-phase water vapor fraction (β_w) prediction correlation plot. The graph shows the target value of the vapor fraction of water from the test data set versus the value of the vapor fraction of water predicted by the neural network. The black dashed line indicates the line of perfect correlation (prediction = target). The data shown has a coefficient of deviation of $R^2 = 0.8821$.

Table 5.18 shows the contribution of misclassification to the value of the mean absolute error for each property in the different phase regions. As can be seen, due to the lower classification accuracy, incorrect classifications contribute a lot more to the total error in the two-phase region than the other two regions.

The error contribution of misclassification being so high for the liquid volume is again due to the fact that the difference in average volume magnitude in the liquid region differs largely from the average magnitudes in the vapor and two-phase regions, so misclassifications lead to costly property mispredictions (similar to what was shown in Figure 5.4).

Table 5.18: Pressure-entropy flash: percentage of the mean absolute error that resulted from instances whose phase was incorrectly classified (ECM). Calculated as per Equation 5.4.

	ECM [%]		
Property	Liquid	Vapor	Two-phase
T	3.71	0.50	40.52
V	17.68	0.02	0.82
H	0.61	0.62	4.99
G	0.96	0.52	22.31
A	1.78	1.40	15.33
U	1.32	0.55	7.91
β_w	-	-	18.25
β_m	-	-	10.29
x_w	-	-	72.83
y_w	-	-	52.17

Entropy, volume flash accuracy scores

Tables 5.19, 5.20 and 5.21 show the test data mean value (μ_{test}) and standard deviation (σ_{test}), and the mean absolute error (MAE), mean absolute error as a percentage of the standard deviation (MAE%), the scaled mean absolute error (MAE$_{\text{sc}}$), and finally the coefficient of deviation (R^2) for the liquid, vapor, and two-phase regression neural networks of the SV-flash.

Table 5.19: Entropy, volume flash: **liquid** region accuracy scores. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). Additionally, the mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
P [Pa]	1.70e+07	8.28e+06	1.57e+07	1.90e+02	5.25e-01	-6.164e+00
T [K]	4.31e+02	9.82e+01	2.04e+00	2.08e+00	4.92e-03	9.976e-01
H [J/mol]	-2.79e+04	1.23e+04	3.61e+02	2.95e+00	6.02e-03	9.986e-01
G [J/mol]	4.77e+03	7.96e+03	2.80e+02	3.51e+00	8.04e-03	9.974e-01
A [J/mol]	4.01e+03	7.59e+03	1.46e+02	1.92e+00	4.34e-03	9.992e-01
U [J/mol]	-2.87e+04	1.18e+04	8.19e+01	6.95e-01	1.45e-03	9.998e-01

Table 5.20: Entropy, volume flash: **vapor** region accuracy scores. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). Additionally, the mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
P [Pa]	1.05e+07	7.78e+06	5.96e+05	7.66e+00	1.99e-02	9.785e-01
T [K]	6.19e+02	6.23e+01	8.78e-01	1.41e+00	2.12e-03	9.995e-01
H [J/mol]	9.72e+03	4.33e+03	1.96e+02	4.53e+00	2.97e-03	9.961e-01
G [J/mol]	1.23e+04	5.44e+03	2.56e+02	4.71e+00	5.39e-03	9.953e-01
A [J/mol]	8.29e+03	5.78e+03	1.63e+02	2.83e+00	3.24e-03	9.984e-01
U [J/mol]	5.67e+03	3.61e+03	1.12e+02	3.10e+00	1.83e-03	9.981e-01

Table 5.21: Entropy, volume flash: **two-phase** region accuracy scores. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). Additionally, the mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
P [Pa]	5.61e+06	4.12e+06	1.80e+06	4.36e+01	6.03e-02	-2.513e+00
T [K]	5.03e+02	5.92e+01	2.74e+00	4.63e+00	6.91e-03	9.732e-01
H [J/mol]	-9.10e+03	8.85e+03	1.36e+02	1.54e+00	2.55e-03	9.989e-01
G [J/mol]	9.89e+03	4.34e+03	2.75e+02	6.33e+00	9.62e-03	9.304e-01
A [J/mol]	8.26e+03	4.43e+03	1.83e+02	4.14e+00	6.30e-03	9.824e-01
U [J/mol]	-1.07e+04	8.11e+03	1.19e+02	1.46e+00	2.38e-03	9.996e-01
β_{w} [mol/mol]	4.58e-01	2.88e-01	1.58e-02	5.48e+00	1.58e-02	9.852e-01
β_{m} [mol/mol]	5.94e-01	2.86e-01	1.78e-02	6.22e+00	1.78e-02	9.810e-01
x_{w} [mol/mol]	7.19e-01	2.09e-01	3.12e-02	1.50e+01	2.55e-02	5.839e-01
y_{w} [mol/mol]	5.77e-01	2.16e-01	4.18e-02	1.93e+01	3.74e-02	4.970e-01

Again, it can be seen that apart from pressure, liquid and vapor region properties have decent MAE and R^2 scores. The reason pressure predictions are very inaccurate in the liquid region is most likely due to the fact that in this region, pressure is very sensitive to changes in volume (and entropy). Figure 5.10 shows a plot of the SV-flash pressure data for all phases together, while Figure 5.11 shows only the liquid region. The second figure shows that in the liquid region pressure ranges from 10^4 to 10^7 Pa, while volume only ranges between 3×10^{-5} m^3/mol to 8×10^{-5} m^3/mol. These large differences in ranges result in very steep gradients, making the resulting curve nearly vertical at times. As was the case for the PT two-phase region, these steep gradients make it very hard to properly train a neural network on the data, which is reflected in the poor accuracy for liquid pressure.

During the development of this work, a few methods of rescaling the liquid pressure were investigated in order to transform the data into a shape which was more conducive to neural network training, but in the end the only viable method could not easily be generalized to new data points, and was therefore not implemented.

The confusion matrix of Table 5.8 in Section 5.2.2 showed that the phase classification network of the SV-flash much more often confused the two-phase region with the liquid region than with the vapor region. This is most likely the due to less data being available near the boundary between the liquid region and the two-phase region, which can be seen from Figure 5.10.

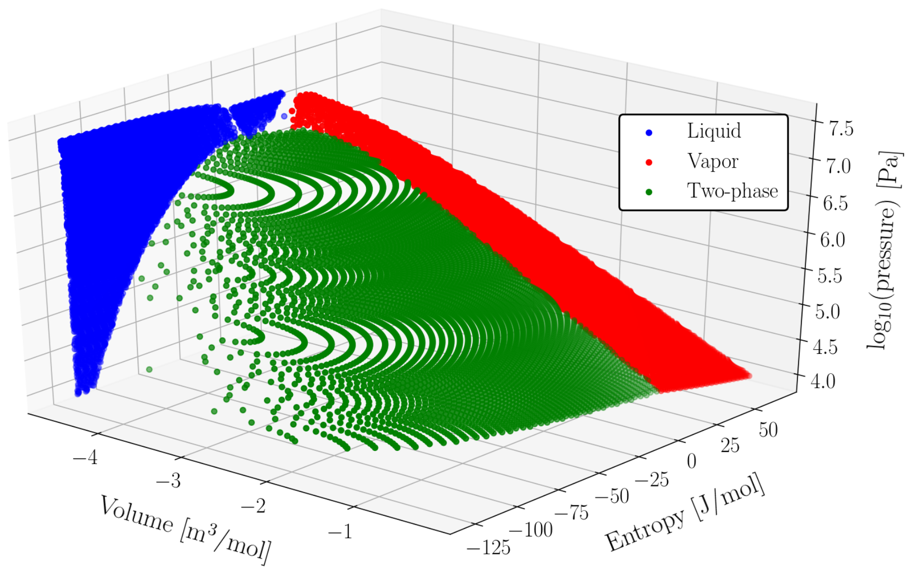


Figure 5.10: Three-dimensional scatter-plot of a part of the data set (single composition) that was used to train the SV-flash pressure regression neural networks. Data is shown for the liquid region (blue), the vapor region (red) and the two-phase region (green). The visual curve-like structures that seem present in the data sets result from the fact that only a certain percentage of all data points are plotted (in order to reduce computational burden on the plotting software).

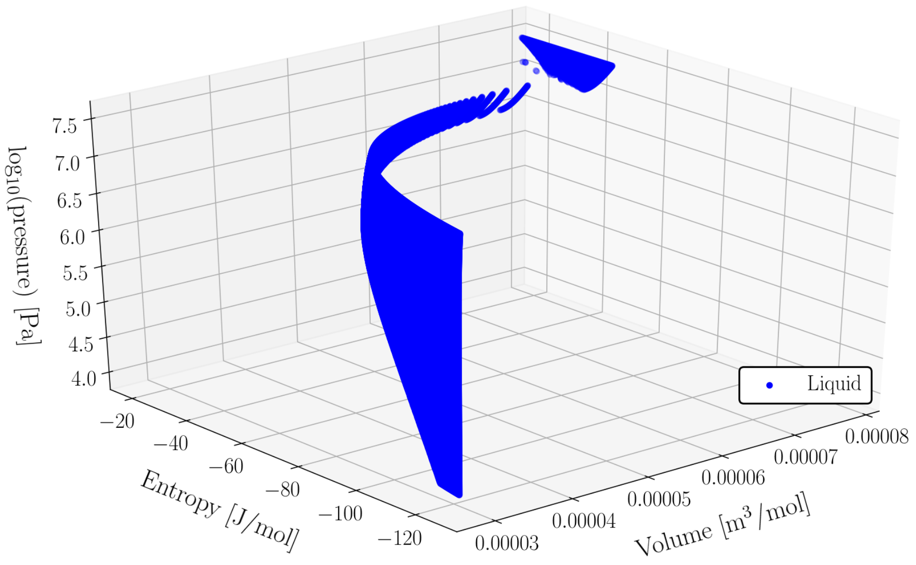


Figure 5.11: Three-dimensional scatter-plot of only the liquid region of a part of the data set (single composition) that was used to train the liquid SV-flash pressure regression neural network. Parts of the data set are so steep, that they look nearly vertical.

As vapor region pressure data showed less steep gradients it resulted in a much higher accuracy when compared to liquid pressure, although it still shows higher error scores when compared to other properties. Two-phase region pressure predictions are also rather inaccurate, this almost completely due to errors in phase classification, which in many cases predicts the liquid region instead of the two-phase region, and as the liquid pressure accuracy is so low, such misclassifications severely impact two-phase region accuracy. This is further reflected by Figure 5.12, which shows the SV-flash correlation plot for two-phase region pressure predictions, and Table 5.22, which shows that for the two-phase region pressure, misclassifications contribute to more than 96% of the total mean absolute error. Most other two-phase properties also show very high misclassification contributions. On the other hand, incorrect classifications barely affect the MAE of properties in the liquid and vapor regions.

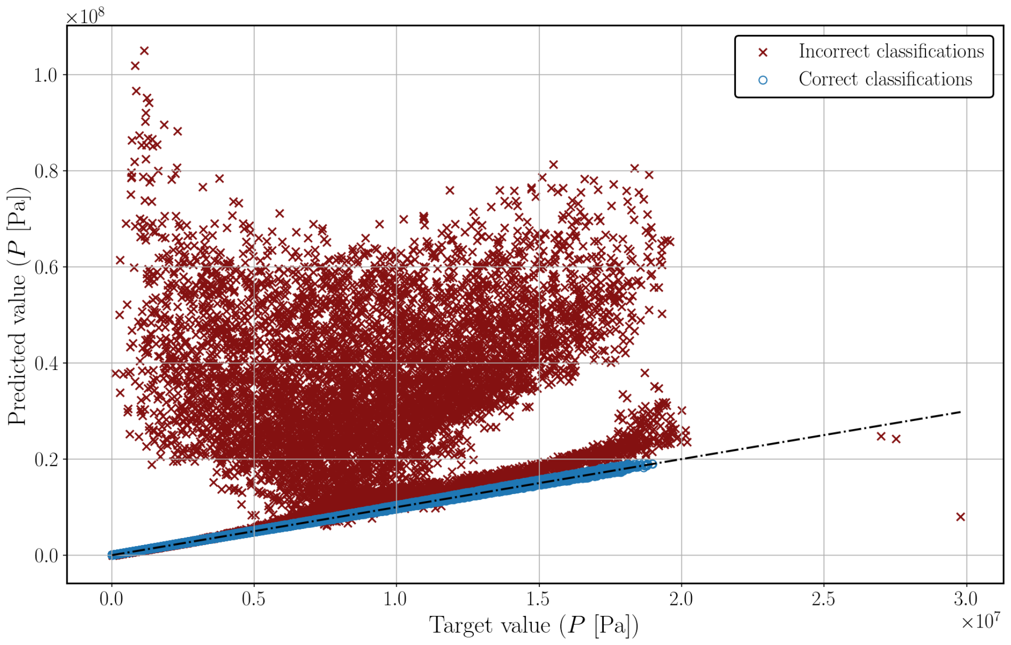


Figure 5.12: Entropy-volume flash: two-phase pressure prediction correlation plot. The graph shows the target value of pressure from the test data set versus the value of pressure predicted by the neural network. The black dashed line indicates the line of perfect correlation (prediction = target). As can be seen, if only correct phase classifications are included, correlation is quite good (R^2 = 0.9996), but incorrect classifications lead to much higher errors overall (R^2 = -2.51).

Table 5.22: Entropy-volume flash: percentage of the mean absolute error that resulted from instances whose phase was incorrectly classified (ECM). Calculated as per Equation 5.4.

	ECM [%]		
Property	Liquid	Vapor	Two-phase
P	0.07	0.01	96.21
T	1.69	0.12	77.00
H	0.16	0.00	46.84
G	0.60	0.10	76.02
A	0.75	0.06	59.05
U	0.78	0.00	3.56
β_{w}	-	-	27.15
β_{m}	-	-	29.72
x_{w}	-	-	63.72
y_{w}	-	-	80.24

Enthalpy, volume flash accuracy scores

Tables 5.23, 5.24 and 5.25 show the test data mean value (μ_{test}) and standard deviation (σ_{test}), and the mean absolute error (MAE), mean absolute error as a percentage of the standard deviation (MAE%), the scaled mean absolute error (MAE$_{\text{sc}}$), and finally the coefficient of deviation (R^2) for the liquid, vapor, and two-phase regression neural networks of the HV-flash.

Table 5.23: Enthalpy, volume flash: **liquid** region accuracy scores. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE_{sc}), and the coefficient of deviation (R^2). Additionally, the mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE_{sc}	R^2
P [Pa]	1.70e+07	8.28e+06	2.01e+07	2.43e+02	6.72e-01	-6.403e+01
T [K]	4.31e+02	9.82e+01	2.03e+00	2.06e+00	4.89e-03	9.985e-01
S [J/mol/K]	-8.17e+01	2.69e+01	8.85e-01	3.29e+00	7.40e-03	9.982e-01
G [J/mol]	4.77e+03	7.96e+03	3.10e+02	3.89e+00	8.91e-03	9.970e-01
A [J/mol]	4.01e+03	7.59e+03	1.51e+02	1.99e+00	4.50e-03	9.993e-01
U [J/mol]	-2.87e+04	1.18e+04	3.13e+02	2.66e+00	5.55e-03	9.989e-01

Table 5.24: Enthalpy, volume flash: **vapor** region accuracy scores. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE_{sc}), and the coefficient of deviation (R^2). Additionally, the mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE_{sc}	R^2
P [Pa]	1.05e+07	7.78e+06	5.13e+05	6.60e+00	1.71e-02	9.824e-01
T [K]	6.19e+02	6.23e+01	1.25e+00	2.01e+00	3.03e-03	9.991e-01
S [J/mol/K]	-4.11e+00	1.28e+01	4.20e-01	3.28e+00	2.27e-03	9.975e-01
G [J/mol]	1.23e+04	5.44e+03	8.58e+01	1.58e+00	1.81e-03	9.996e-01
A [J/mol]	8.29e+03	5.78e+03	1.82e+02	3.16e+00	3.62e-03	9.980e-01
U [J/mol]	5.67e+03	3.61e+03	1.15e+02	3.19e+00	1.88e-03	9.980e-01

Table 5.25: Enthalpy, volume flash: **two-phase** region accuracy scores. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE_{sc}), and the coefficient of deviation (R^2). Additionally, the mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE_{sc}	R^2
P [Pa]	5.61e+06	4.12e+06	5.08e+05	1.23e+01	1.71e-02	7.199e-01
T [K]	5.03e+02	5.92e+01	3.08e+00	5.21e+00	7.78e-03	9.873e-01
S [J/mol/K]	-3.81e+01	1.81e+01	5.36e-01	2.97e+00	3.96e-03	9.984e-01
G [J/mol]	9.89e+03	4.34e+03	3.27e+02	7.53e+00	1.14e-02	9.867e-01
A [J/mol]	8.26e+03	4.43e+03	2.12e+02	4.79e+00	7.30e-03	9.943e-01
U [J/mol]	-1.07e+04	8.11e+03	2.59e+02	3.20e+00	5.20e-03	9.984e-01
β_{w} [mol/mol]	4.58e-01	2.88e-01	1.34e-02	4.66e+00	1.34e-02	9.768e-01
β_{m} [mol/mol]	5.94e-01	2.86e-01	1.81e-02	6.32e+00	1.81e-02	9.761e-01
x_{w} [mol/mol]	7.19e-01	2.09e-01	3.63e-02	1.74e+01	2.88e-02	4.519e-01
y_{w} [mol/mol]	5.77e-01	2.16e-01	3.87e-02	1.79e+01	3.56e-02	5.385e-01

Results from the HV-flash strongly mirror the results of the SV-flash: overall decent to good accuracy in the liquid and vapor regions, apart from pressure predictions, which are inaccurate for the same reasons as given for the SV-flash. Again, two-phase property predictions are overall lower due to less training data being available and incorrect classifications leading to incorrect value predictions. Table 5.26 further shows the large contributions to the total mean absolute error of two-phase region properties that result from incorrect phase classifications.

Table 5.26: Enthalpy-volume flash: percentage of the mean absolute error that resulted from instances whose phase was incorrectly classified (ECM). Calculated as per Equation 5.4.

	ECM [%]		
Property	Liquid	Vapor	Two-phase
P	0.00	0.08	89.31
T	0.00	0.06	49.17
S	0.00	0.02	11.81
G	0.00	0.04	16.37
A	0.00	0.03	25.25
U	0.00	0.01	11.47
β_w	-	-	50.64
β_m	-	-	37.47
x_w	-	-	78.32
y_w	-	-	77.05

5.3 Neural network robustness

Robustness refers to the tendency of a calculation method to return valid answers. Non-valid answers can result from the calculation method not converging, converging to an incorrect answer, or otherwise crashing. In addition, in this text robustness will also refer to the ability of a neural network to still provide good predictive capabilities in the presence of gaps or inconsistencies within a training data set.

5.3.1 Robustness to failure

The first sense of the meaning of robustness is relatively straightforward for a neural network: a properly trained neural network given valid inputs will return a valid output in almost 100% of the cases. Going from input to output in a neural network only requires matrix multiplication, matrix addition, and application of an activation function which in the most complicated cases only contains a logarithm or exponent. In general, the only ways in which a neural network may fail include:

- An activation function containing a logarithm or exponent can return value of `NaN` or ± `inf` when it is given an input with a value of zero or less than zero. However, most widely used activation functions that contain logarithms and exponents (such as the logistic sigmoid or tanh functions) are valid over the entire range of real numbers.

- If the number of input combinations given to the neural network at once is too large, the resulting matrices could be too large to fit in the computers memory, inevitably leading to the code (or entire computer) crashing. As this type of failure is due to hardware limitations, the failure should not be attributed to the neural network itself.

5.3.2 Robustness to incorrect/incomplete data

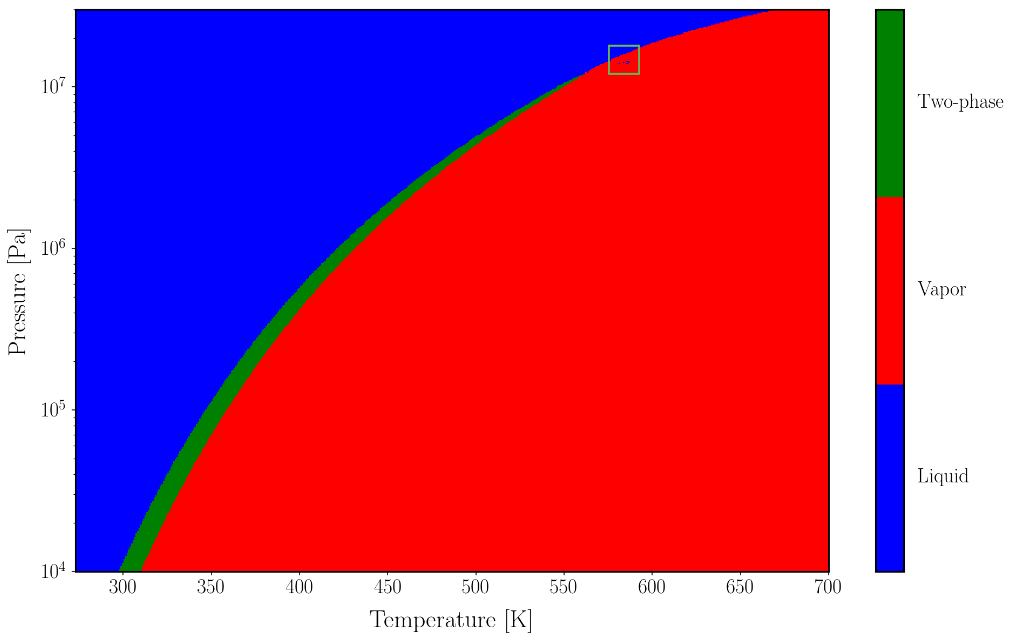
In this subsection robustness in the second sense of the word is examined. Two examples will be given of artificial neural networks that were trained on data sets that either included incorrect target values, or had gaps in the training data.

Robustness to incorrect data

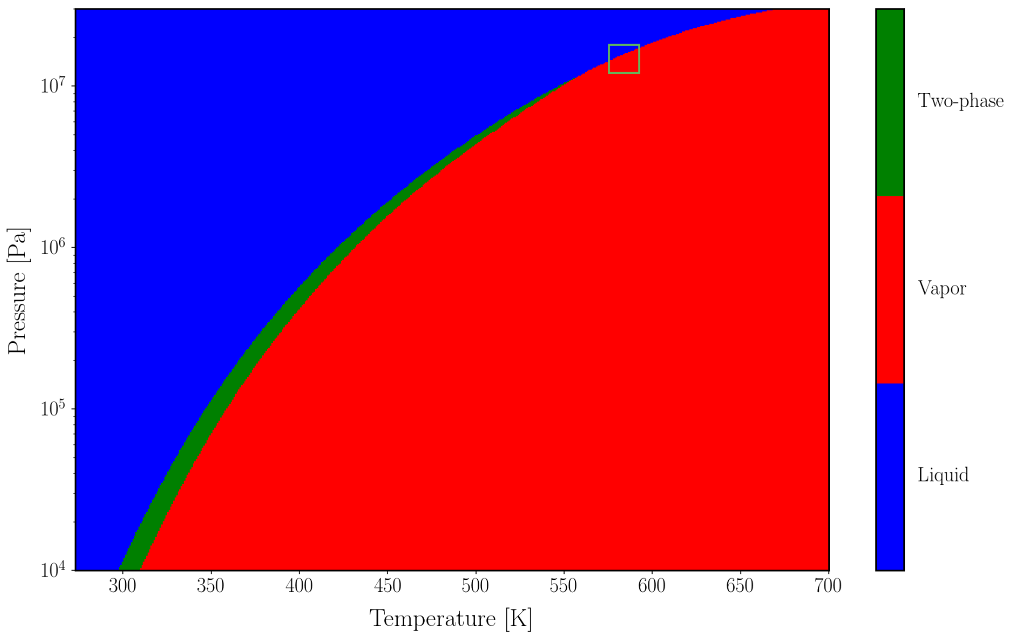
Figure 5.13a shows a pressure temperature phase plot of a 50-50 mol% water-methanol mixture. As can be seen, the plot shows a few errors in the classification between the liquid and vapor phase in the box near the top-right corner of the plot, these resulted from a failure in TEAs algorithm. Instead of discarding these errors or manually correcting them, they were kept as they were. Figure 5.14a shows a close-up view of the section in question.

Figure 5.13b shows a similar pressure temperature phase plot based on predictions made by an artificial neural network that was trained on parts of the data shown in Figure 5.13a. Figure 5.14b shows the same section as Figure 5.14a. From both sections plots, it can be seen that the neural network predictions don't include the incorrect predictions that were present in the original data from TEA. The reason for this is that the neural network tries to minimize the overall error, while in theory it "wants" to include the incorrect points (since according to the training data the network is given they are correct), doing so would lead it to incorrectly classify even more points than if it wouldn't (since it would incorrectly classify the surrounding points). Thus, in order to minimize the total error, the best thing the network can do is not include the original outliers, which actually turns out to be beneficial to the real accuracy of the network.

The above example nicely shows how neural networks can predict correct outcomes even if incorrect data is present in the original data set it is trained with, however, only few incorrect data point were present. In the next section, a numerical experiment is performed in which much larger parts of a data set are omitted.

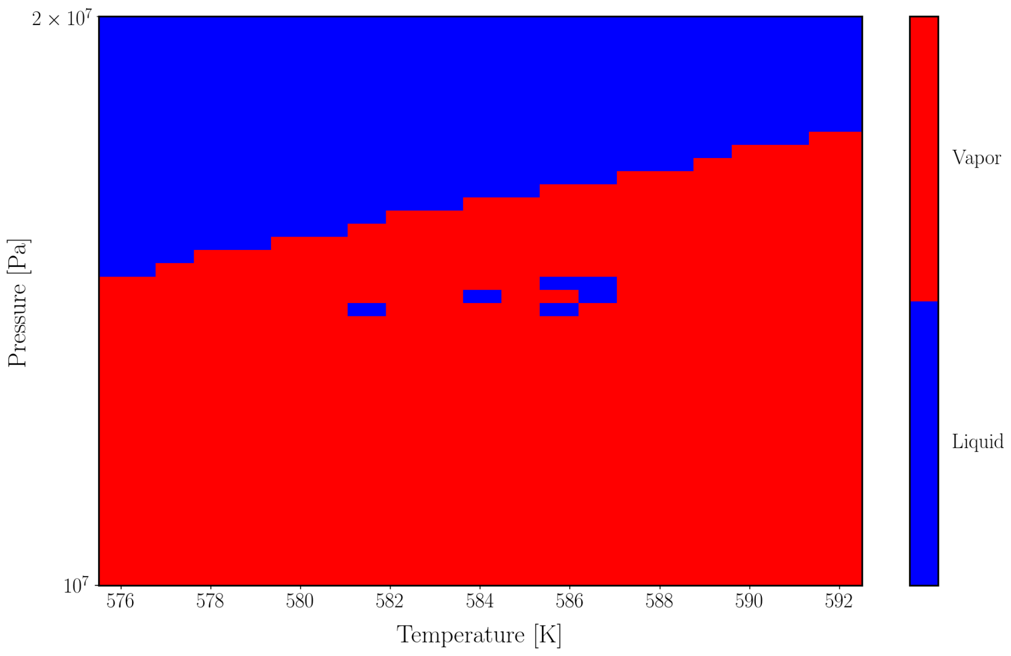


(a)

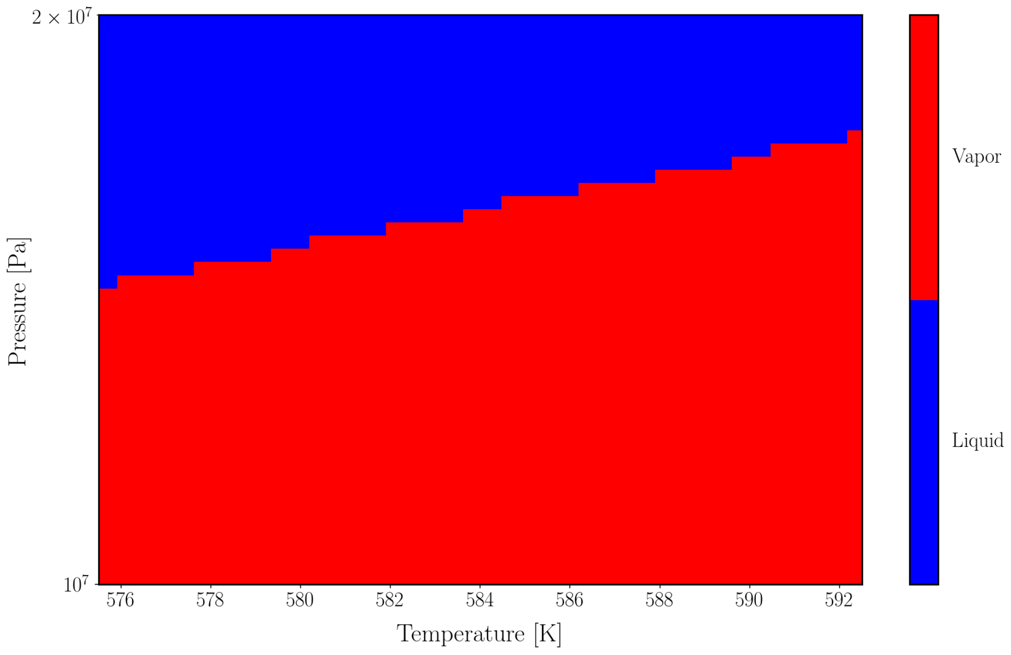


(b)

Figure 5.13: Pressure temperature phase plots of a 50-50 mol% mixture of water and methanol. Data generated using the TEA *PT*-flash algorithm [11, 31] (a) and a trained artificial neural network (b). The data generated with TEA shows some errors in the top-right corner (light-green box) that the neural network does not show. A close-up section view of the area in question is shown in Figure 5.14.



(a)



(b)

Figure 5.14: Section views of the plots of Figure 5.13a (a) and Figure 5.13b (b). As can be seen, there are a few instances in (a) where the TEA algorithm [11,31] predicts a liquid phase where it should predict a vapor phase. The ANN predictions plotted in (b) show no such errors.

Robustness to incomplete data

Figures 5.15 and 5.16 show respectively the temperature-density and pressure-density coexistence curves for pure methane as calculated through an equation of state (green) and as predicted by a neural network (blue) which was trained on only a part of the example data set (shown in red). As can be seen, the neural network can still fairly accurately predict the missing values, even when trained on a data set from which 30% was left out near the critical point. Appendix G gives a more detailed overview of how these results were obtained.

This example shows that even when data points are missing from a training data set, neural networks can still predict to a relatively high accuracy values of points in the region where no training data was available.

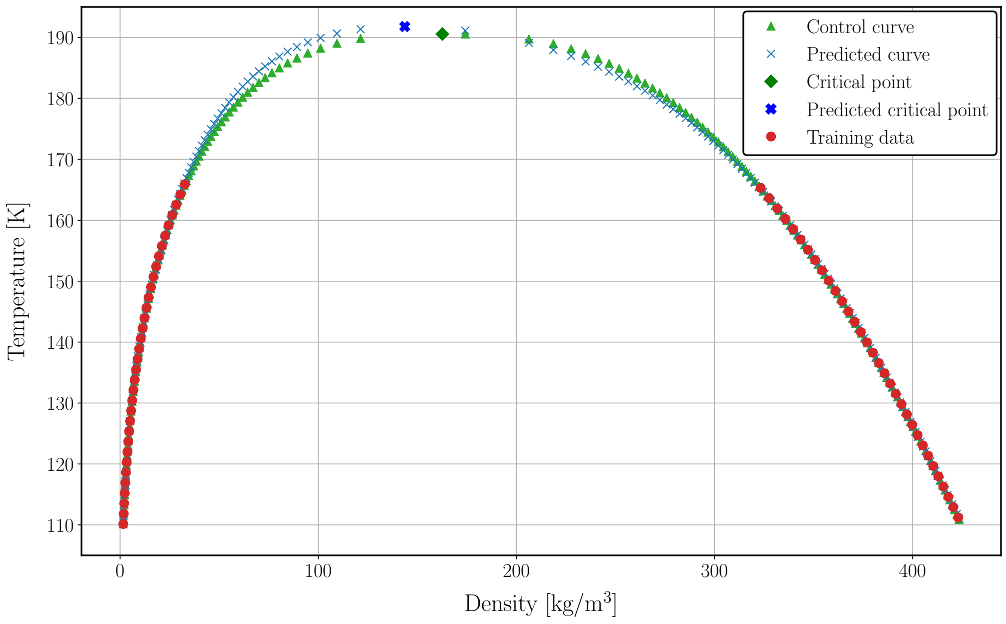


Figure 5.15: Temperature-density coexistence curve for pure methane, comparison between the complete target data set (green) and predicted using a neural network with three hidden neurons (blue). Red points show (a selection of) the data points on which the neural network was trained.

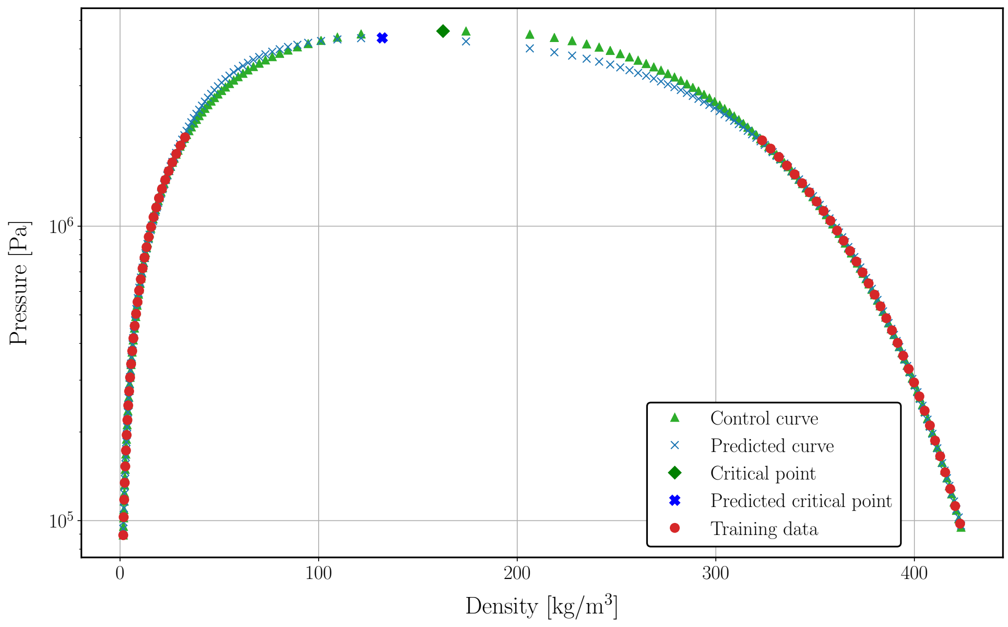


Figure 5.16: Pressure-density coexistence curve for pure methane, comparison between the complete target data set (green) and predicted using a neural network with three hidden neurons (blue). Red points show (a selection of) the data points on which the neural network was trained.

5.4 Thermodynamic consistency

There are many thermodynamic expressions which relate (derivatives of) one state variable to another, any good thermodynamic model will need to be consistent with these relations in order to be considered a good replacement for conventional methods. In this section, the artificial neural network PS-flash is used to determine a couple of state function derivatives, which are then checked to which extent they satisfy relevant thermodynamic relations. The PS-flash was chosen because overall it has the highest accuracy scores, and contains no phase region(s) for which it struggles to accurately predict certain properties.

5.4.1 Thermodynamic consistency: methods

The main relation that will be checked are those relating to the fundamental equation $H(P, S)$. As mentioned in Chapter 2, by taking the derivatives of the fundamental equation with respect to its natural variables, all other thermodynamic properties can be derived. More specifically, the neural network PS-flash will be used to predict the following relations:

$$\left(\frac{\partial H}{\partial P}\right)_S = V \tag{5.5a}$$

$$\left(\frac{\partial H}{\partial S}\right)_P = T \tag{5.5b}$$

$$\left(\frac{\partial T}{\partial P}\right)_S = \left(\frac{\partial V}{\partial S}\right)_P \tag{5.5c}$$

Where the last expression is one of the Maxwell relations [14]. All derivatives are determined though numerical differentiation using a two-point central difference method:

$$\frac{\partial f}{\partial x} \approx \frac{f(x+\delta x) - f(x-\delta x)}{2\delta x} \tag{5.6}$$

The value of δx was varied for each derivative until no more increase in accuracy could be achieved. Results from the numerical derivatives will be compared to their true values found in the the same test data sets as used in Section 5.2. The Maxwell partial derivatives will be compared to each other. Error metrics will be same as used in Section 5.2.

5.4.2 Thermodynamic consistency: results

Derivative of enthalpy with respect to pressure

Table 5.27 shows the mean value and standard deviation of the target volume data set and accuracy scores of the numerical derivatives of enthalpy with respect to pressure. In addition, Figures 5.17, 5.18, and 5.19 show the correlation plots of each phase region.

Both the table and figures show that while for the vapor and two-phase regions the derivative fairly accurately predicts the real value, in the liquid region this is far from the case. A likely reason for this result is that liquid volumes are generally very small, which makes approximating them with a numerical derivative more difficult. The low R^2 score for the two-phase region is skewed due to the presence of very pronounced outliers, Figure 5.19 shows that if these outliers are disregarded, correlation is much better.

Moreover, results from Section 5.2.3 show that direct predictions of volume using a neural network are for all phase regions more accurate than through the numerical derivative of enthalpy. For example, the vapor volume predicted directly with a neural network gives an R^2-score of 0.999, while calculating it through numerical differentiation of enthalpy gives an R^2-score of 0.992.

It should also be noted that out of all 300,000 predictions made for all phase regions combined, 399 negative values for volume were predicted through the numerical derivative of enthalpy with respect to pressure: 267 in the liquid region, 1 in the vapor region, and 131 in the two-phase region. While these number are quite low when compared to the entire size of the data set, any negative volume prediction is a serious violation of the laws of thermodynamics, which should never occur. If this problem cannot be solved, thermodynamic consistency can not be achieved, no matter how accurate the remaining predictions are.

Table 5.27: Accuracy scores of the numerical derivative of enthalpy predicted using the artifical neural network with respect to pressure at constant entropy compared to actual volume accuracy scores. The mean value and standard deviation shown are those for the volume targets data set.

Phase region	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
Liquid	4.19e-05	2.25e-05	2.14e-05	9.52e+01	2.0e-01	-9.48e+00
Vapor	2.50e-03	1.37e-02	1.39e-04	1.01e+00	2.61e-04	9.92e-01
Two-phase	1.24e-03	5.39e-03	8.24e-05	1.53e+00	4.16e-04	7.99e-01

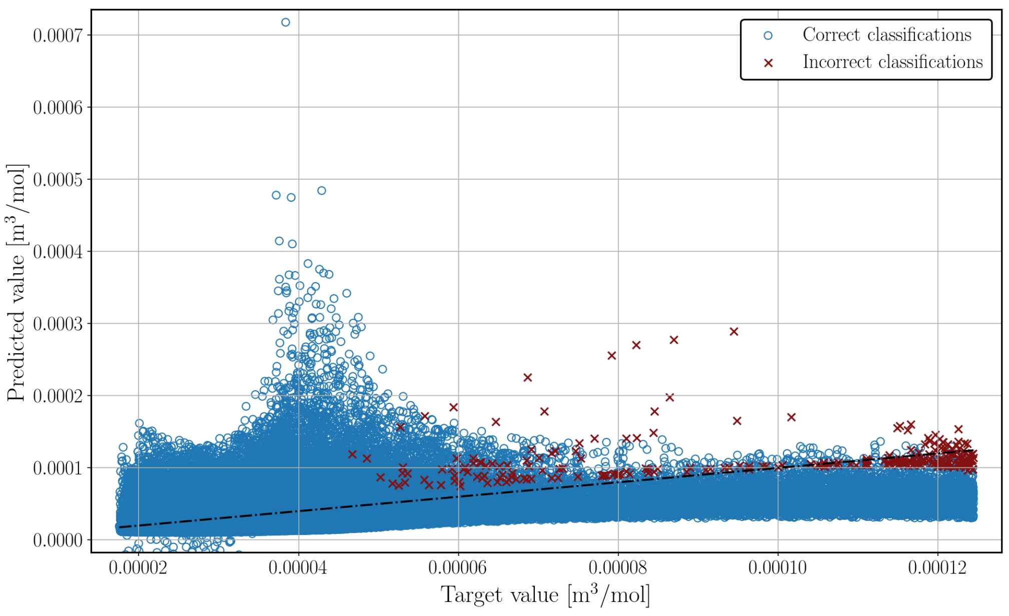


Figure 5.17: Liquid region volume predictions using the numerical derivative of neural network predicted values of enthalpy with respect to pressure at constant entropy versus the actual volume correlation plot. Blue circles represent instances that were correctly classified as being liquid, while the red crosses indicate incorrect classifications.

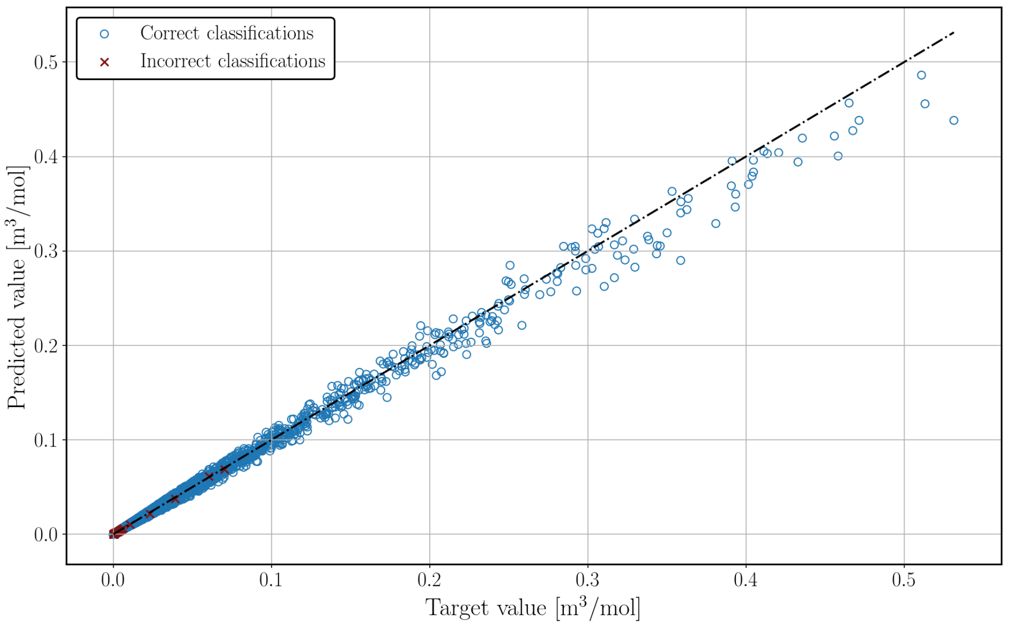


Figure 5.18: Vapor region volume predictions using the numerical derivative of neural network predicted values of enthalpy with respect to pressure at constant entropy versus the actual volume correlation plot. Blue circles represent instances that were correctly classified as being vapor, while the red crosses indicate incorrect classifications.

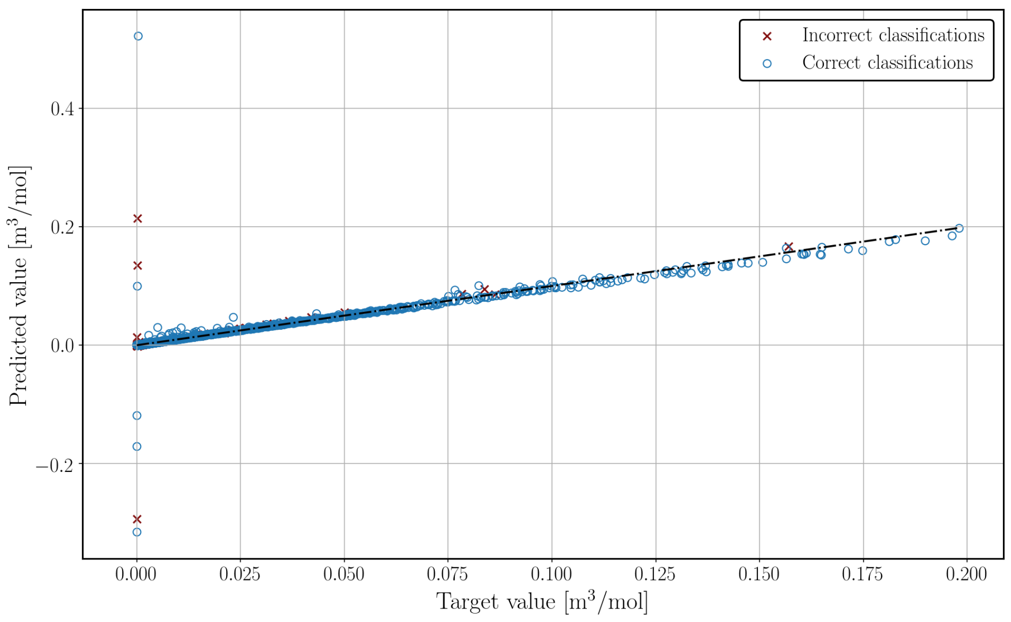


Figure 5.19: Two-phase region volume predictions using the numerical derivative of neural network predicted values of enthalpy with respect to pressure at constant entropy versus the actual volume correlation plot. Blue circles represent instances that were correctly classified as being two-phase, while the red crosses indicate incorrect classifications.

Derivative of enthalpy with respect to entropy

Table 5.28 shows the mean value and standard deviation of the target temperature data set and accuracy scores of the numerical derivatives of enthalpy with respect to entropy. In addition, Figures 5.20, 5.21, and 5.22 show the correlation plots of each phase region.

These results show less accurate predictions as the previous subsection, only the liquid region results are somewhat decent, but again results obtained through direct neural network predictions are more accurate for all phase regions. Figure 5.22 again shows the effect incorrect phase classifications can have on the results from property value predictions.

Table 5.28: Accuracy scores of the numerical derivative of enthalpy predicted using the artifical neural network with respect to entropy at constant pressure compared to actual temperature. The mean value and standard deviation shown are those for the volume targets data set.

Phase region	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
Liquid	4.31e+02	9.82e+01	5.08e+00	5.17e+00	1.23e-02	9.94e-01
Vapor	6.19e+02	6.23e+01	2.79e+01	4.48e+01	6.75e-02	6.86e-01
Two-phase	5.03e+02	5.92e+01	1.15e+01	1.94e+01	2.90e-02	9.24e-01

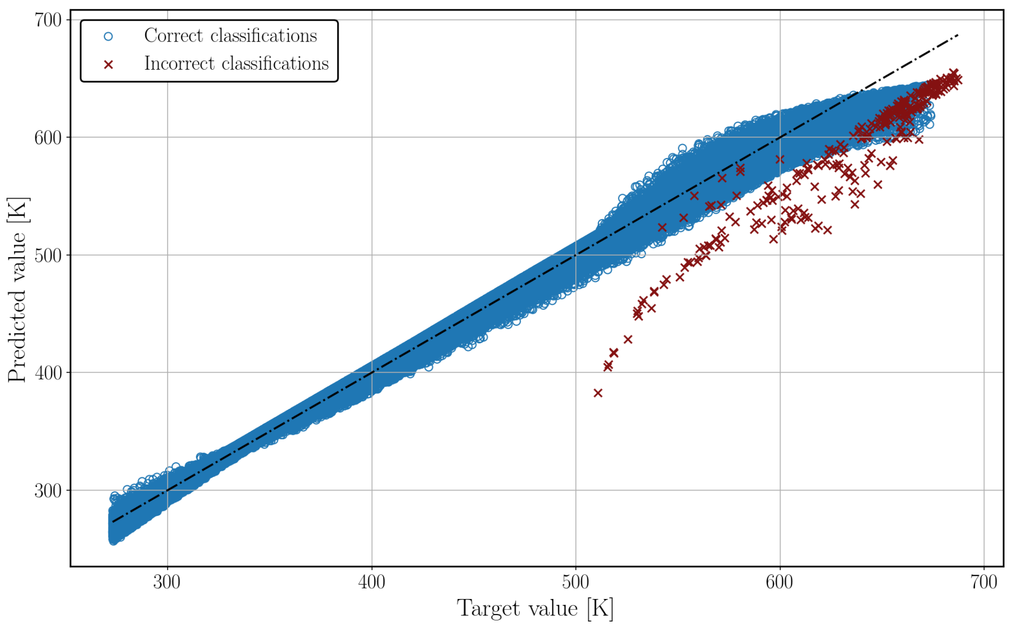


Figure 5.20: Liquid region temperature predictions using the numerical derivative of neural network predicted values of enthalpy with respect to entropy at constant pressure versus the actual temperature correlation plot. Blue circles represent instances that were correctly classified as being liquid, while the red crosses indicate incorrect classifications.

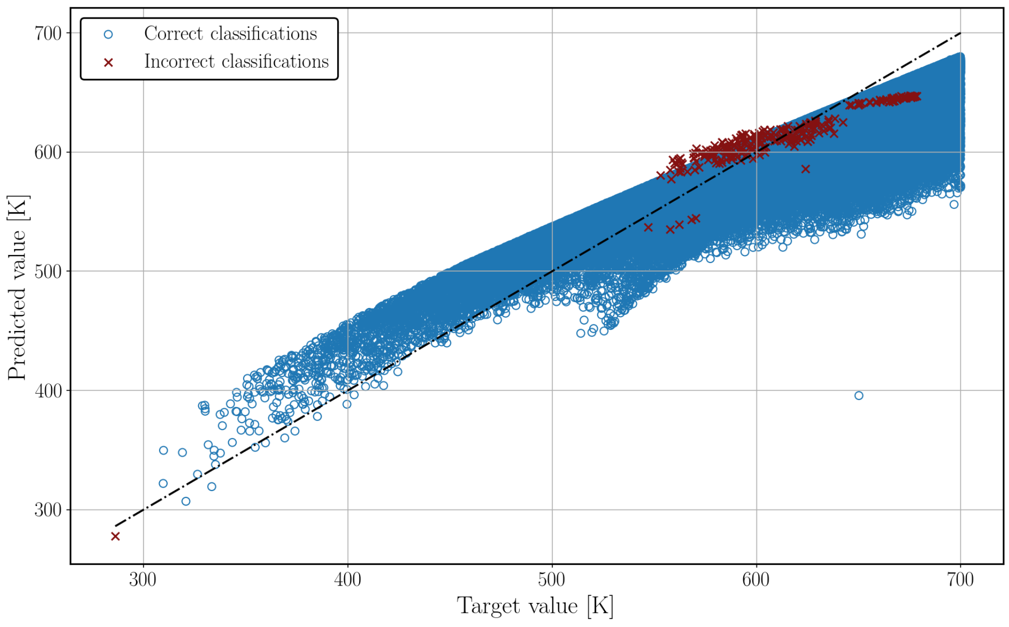


Figure 5.21: Vapor region temperature predictions using the numerical derivative of neural network predicted values of enthalpy with respect to entropy at constant pressure versus the actual temperature correlation plot. Blue circles represent instances that were correctly classified as being vapor, while the red crosses indicate incorrect classifications.

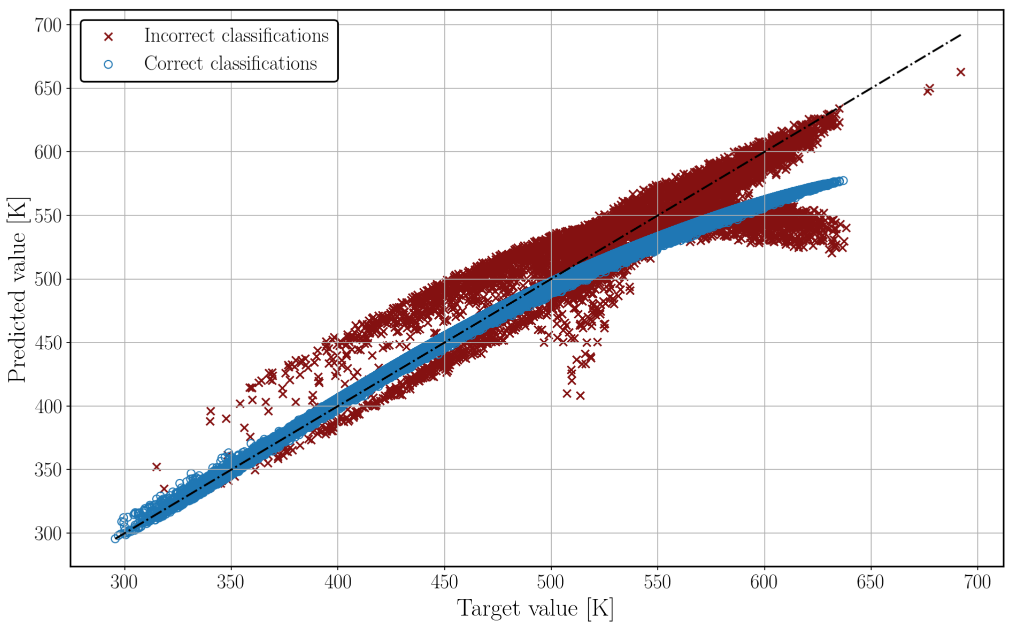


Figure 5.22: Two-phase region temperature predictions using the numerical derivative of neural network predicted values of enthalpy with respect to entropy at constant pressure versus the actual temperature correlation plot. Blue circles represent instances that were correctly classified as being two-phase, while the red crosses indicate incorrect classifications.

Maxwell relation

Table 5.29 shows the mean value and standard deviation of the numerical derivatives of temperature with respect to pressure, and accuracy scores of the numerical derivatives of volume with respect to entropy compared to the former derivative. In addition, Figures 5.23, 5.24, and 5.25 show the correlation plots between the two derivatives of each phase region.

The results show a similar pattern as for the derivative of enthalpy with respect to pressure: decent accuracy in the vapor and two-phase regions, and very poor results for the liquid region. These results clearly show that the neural network PS-flash does not satisfy the maxwell relation.

Table 5.29: Accuracy scores of numerical derivative of temperature predicted using the artifical neural network with respect to pressure at constant entropy (x-axis) compared to numerical derivative of volume predicted with the same neural network with respect to entropy at constant pressure (y-axis). The mean value and standard deviation shown are those for the first numerical derivative.

Phase region	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
Liquid	5.14e-07	8.16e-07	4.31e-07	5.29e+01	4.29e-03	-4.27e-01
Vapor	5.68e-05	2.92e-04	6.16e-06	2.11e+00	4.76e-04	9.88e-01
Two-phase	2.83e-05	7.80e-05	2.01e-06	2.58e+00	5.49e-04	9.88e-01

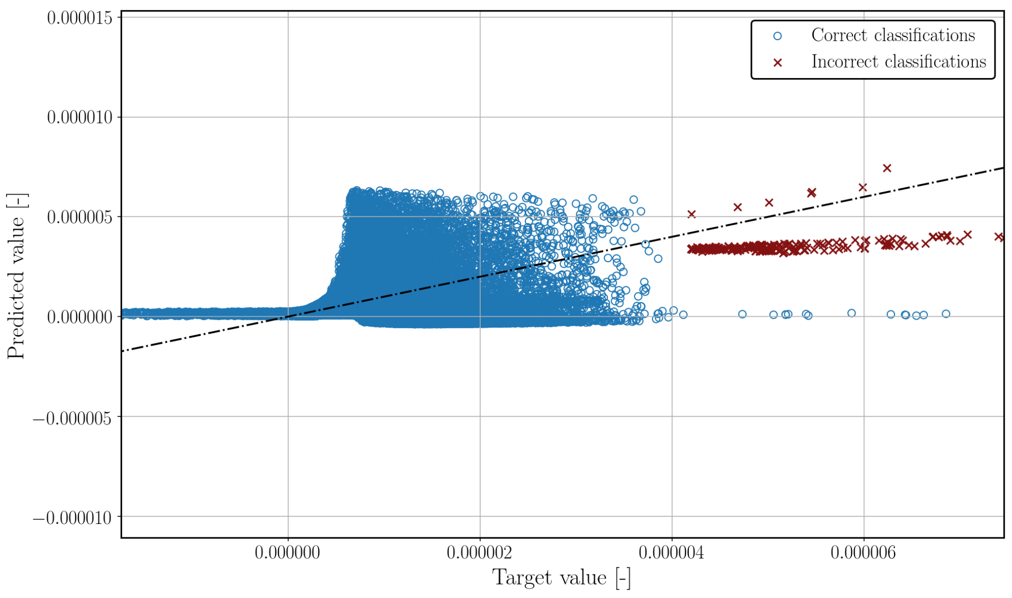


Figure 5.23: Liquid region correlation plot comparing numerical derivative of temperature with respect to pressure at constant entropy to the numerical derivative of volume with respect to entropy at constant pressure based on prediction made with the artificial neural network. Blue circles represent instances that were correctly classified as being liquid, while the red crosses indicate incorrect classifications.

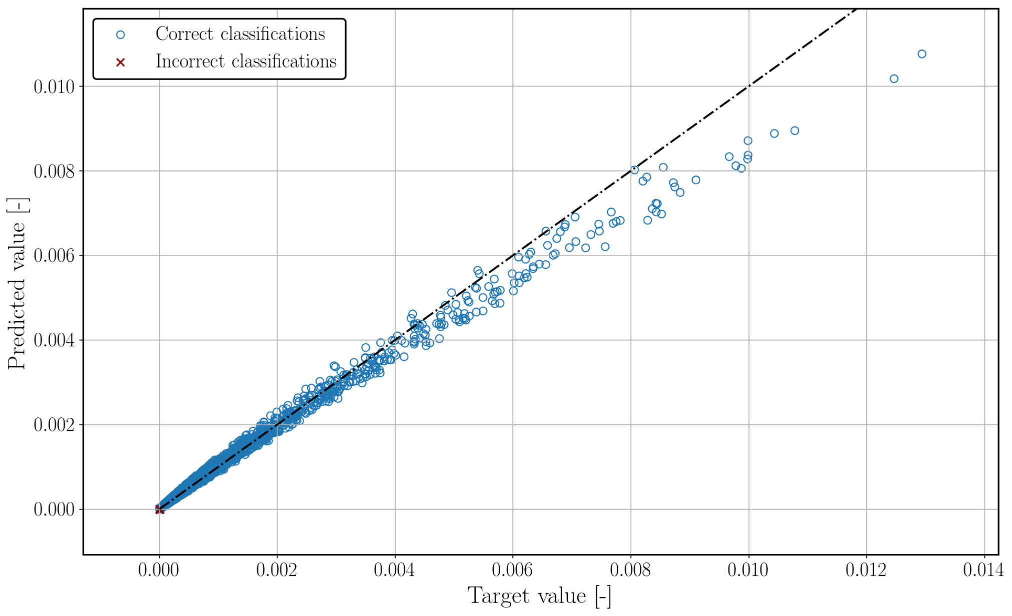


Figure 5.24: Vapor region correlation plot comparing numerical derivative of temperature with respect to pressure at constant entropy to the numerical derivative of volume with respect to entropy at constant pressure based on prediction made with the artificial neural network. Blue circles represent instances that were correctly classified as being vapor, while the red crosses indicate incorrect classifications.

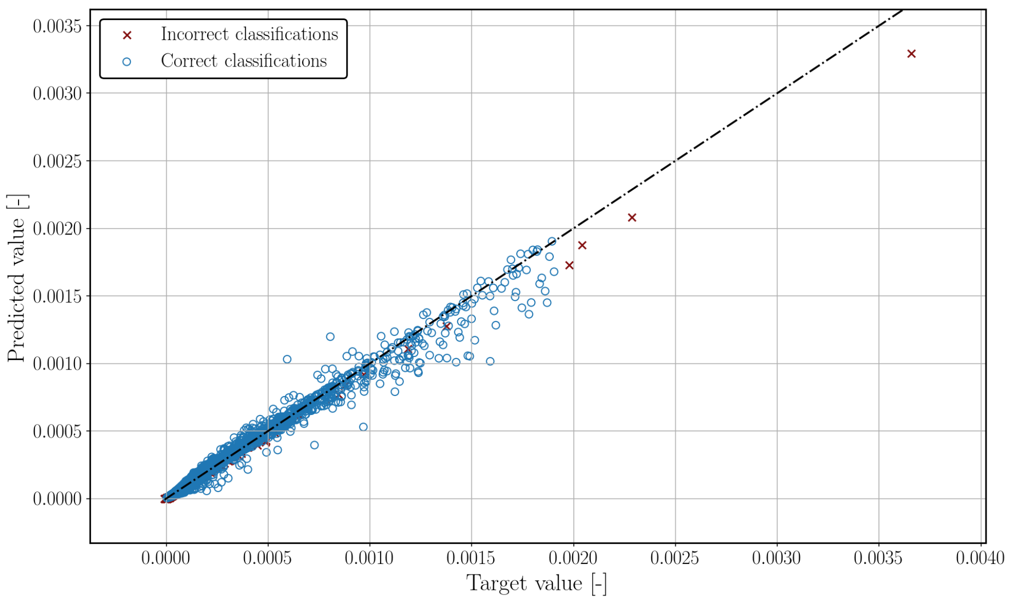


Figure 5.25: Two-phase region temperature predictions using the numerical derivative of neural network predicted values of enthalpy with respect to entropy at constant pressure versus the actual temperature correlation plot. Blue circles represent instances that were correctly classified as being two-phase, while the red crosses indicate incorrect classifications.

5.5 Stability of a system of flashes

The work accomplished in this project has resulted in four artificial neural network flash types: a PT-, PS-, SV-, and HV-flash. In this section the stability of a system composed of two of these flashes will be investigated.

In the context of this thesis, stability is defined as the tendency of a system to deviate from it's initial state if no (or only very slight) changes occur within the system. A stable system is one whose initial property values differ very little, or not at all, from the values they attain after a relatively large number of iterations in which no changes are induced by external actors.

A more concrete example is illustrated using Figure 5.26. The figure shows a system containing a PT-flash and PS-flash. The system starts at an initial state with values P^0, T^0 (and S^0), based on the initial values of pressure and temperature, the PT-flash predicts a new value for entropy S. The new value of entropy and the unchanged value of pressure are used by the PS-flash to predict a new temperature value T, which is in turn given to the PT-flash to again predict a new value for entropy. This cycle is repeated for a number of iterations, after which the values of temperature and entropy at the final state are compared to their values at the initial state, how much the values differ is used as a measure of stability of the system.

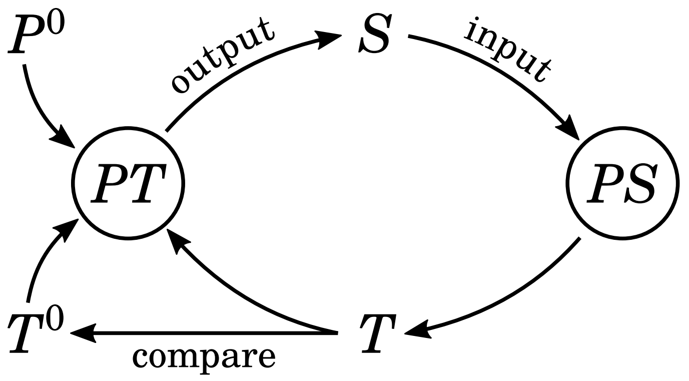


Figure 5.26: Schematic overview of an example stability test. Initial pressure and temperature (and composition) are given as inputs to the PT-flash. The PT-flash is used to predict a value for entropy (S), which is given as an input (along with the constant values of pressure and composition) to the PS-flash. The PS-flash predicts a new value for temperature (T), which is again given to the PT-flash. This cycle is repeated a number of times, after which the final values of temperature and entropy are compared to their initial values. The difference between final and initial values gives an indication of the stability of the system.

5.5.1 System stability: methods

Stability was investigated based on the approach discussed at the start of this section. The neural network based PT- and PS-flashes were cycled for 500 iterations while keeping pressure and composition constant, for each iteration the values of temperature and entropy were recorded. At the end of the 500 iterations the difference in value from the initial state was saved, and it was checked which of the following three final states of the system was reached:

- *Stable.* The system has reached a constant value of temperature that will result in a constant value of entropy, no changes have occurred for at least 50 iterations.
- *Diverging.* The system has not yet reach a stable point after 500 iterations, values of temperature and entropy are still increasing or decreasing.
- *Limit cycle.* The system has not reached a stable singular state yet, but is also not monotonically increasing or decreasing. Instead, values of temperature and entropy have been cycling through a limited set of values for at least the last 50 iterations.

Iterations were repeated for 5000 randomly chosen starting values (P^0, T^0, and z^0) in the liquid, vapor, and two-phase regions (15000 in total).

5.5.2 System stability: results

Liquid region

Table 5.30 shows the main results for the liquid region stability analysis. As can be seen, on average the value of temperature after 500 iterations differs around 70 K from the value it started at, for entropy this value is around 20 J/mol/K. In addition, for around 7% of cases the phase at the end state was not the same as that for the beginning state. One good thing is that almost 90% of the times a system will at least reach a stable value, never diverges for more than 500 iterations, and reaches a limit cycle in only around 12 % of the times.

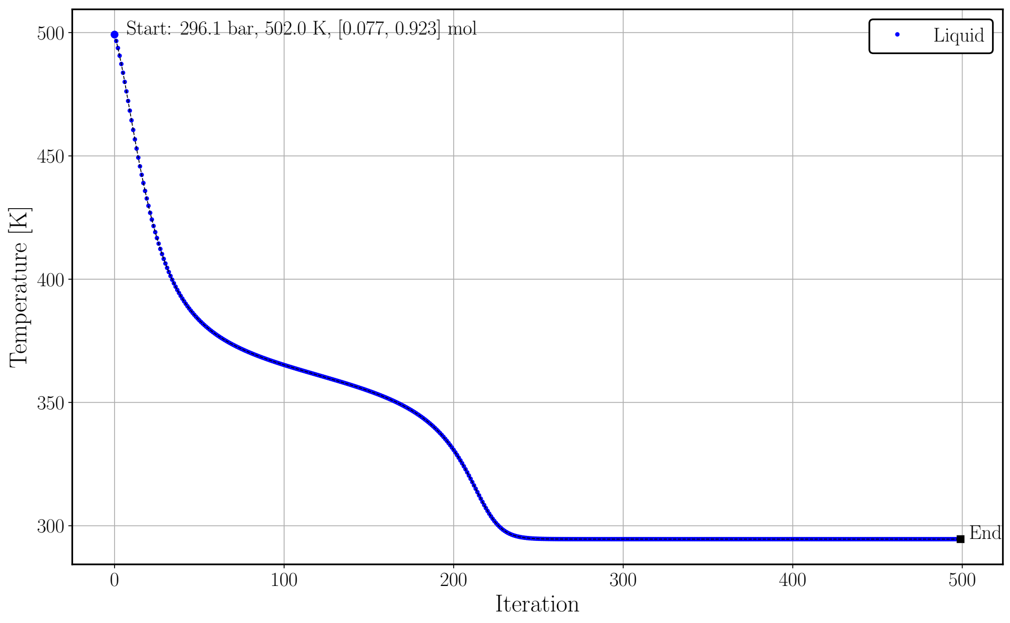
Figures 5.27a and 5.27b show plots of temperature from the PS-flash and entropy from the PT-flash against the iteration number for a system that reaches a stable point after around 300 iterations. Figure 5.28a and 5.28b show plots of temperature from the PS-flash and entropy from the PT-flash against the iteration number for a system that reaches a limit cycle at around the 25 iterations mark.

For all situation in which a limit cycle was reached and plots such as Figure 5.28a and 5.28b were available, the plots showed that part of the limit cycle included the system being classified as a different phase from the one it started at for one iteration, before again being classified as its original phase in the next iteration. From these results a possible explanation for how the limit cycles are formed can be derived: the system starts of by converging towards a certain value, but at one point encounters a value for which the system is incorrectly classified as another phase. In the next iteration the system is again correctly classified as its original phase, but is now at a different point with a value lower than the original value it was converging towards. The system thus resumes converging until it again reaches the value at which it was originally incorrectly classified, and the cycle repeats.

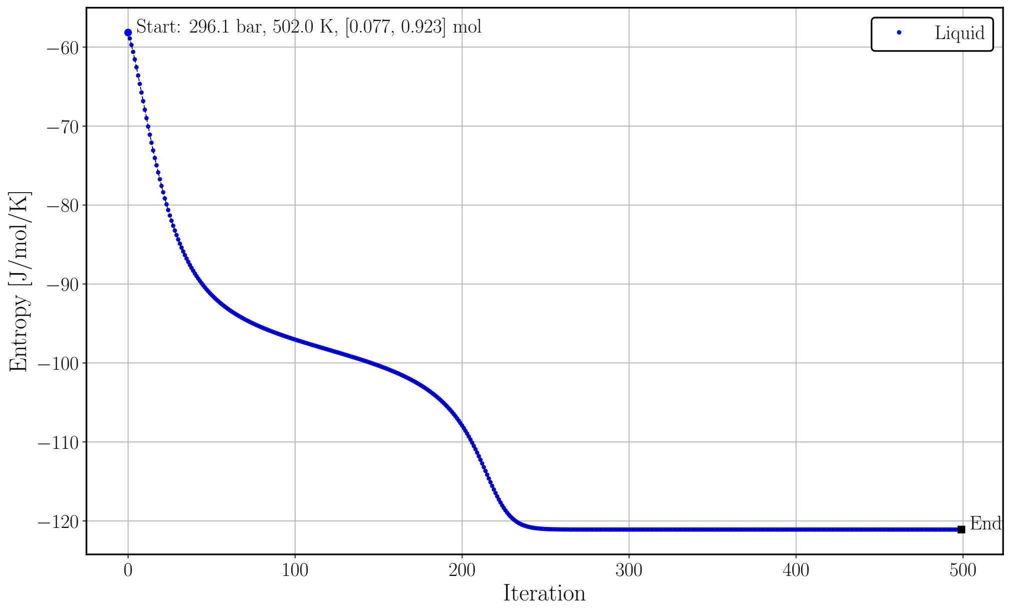
From the table and figures it should be clear that the systems made up of the neural network PT- and PS-flashes are in most cases unstable, as their values of temperature and entropy at the final state are on average quite far removed from the values at their initial state. A silver lining is that at least a large majority of systems eventually reach some point of constant state (be it often times far removed from the intitial one).

Table 5.30: Liquid region stability analysis results for the PT-PS-flash system. Mean temperature and entropy deviations indicate the difference between the value of the property at the first iteration and at the last iteration. Percentages of the number properties indicate in how many of the total cases it occurred.

Metric	Value	
Mean temperature deviation	68.6	[K]
Mean entropy deviation	19.9	[J/mol/K]
Number of phase transition	360	(7.2 %)
Number of converging systems	4409	(88.2 %)
Number of diverging systems	0	(0.0 %)
Number of limit cycle systems	591	(11.8 %)

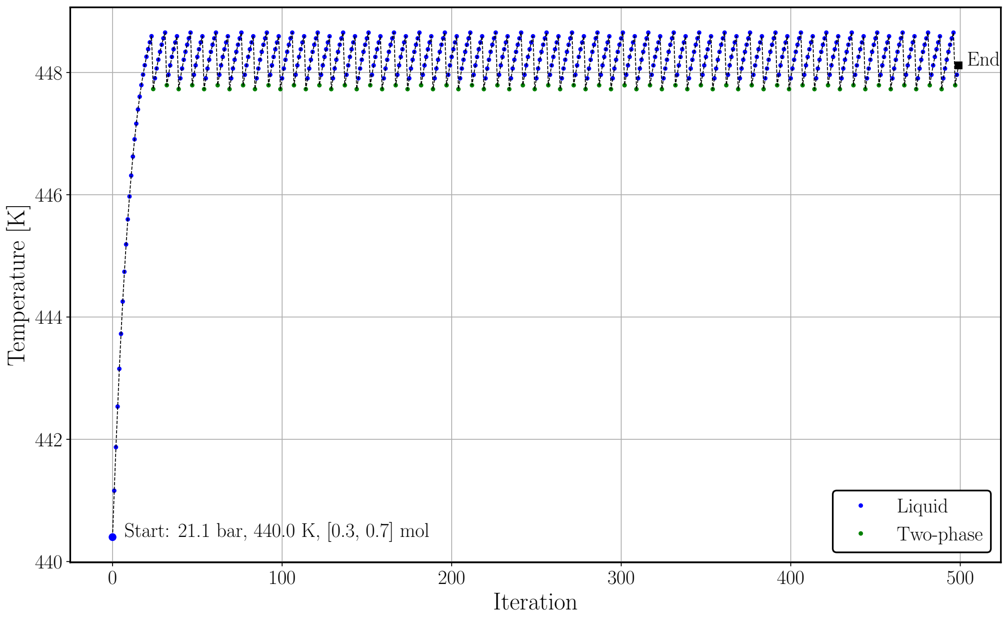


(a)

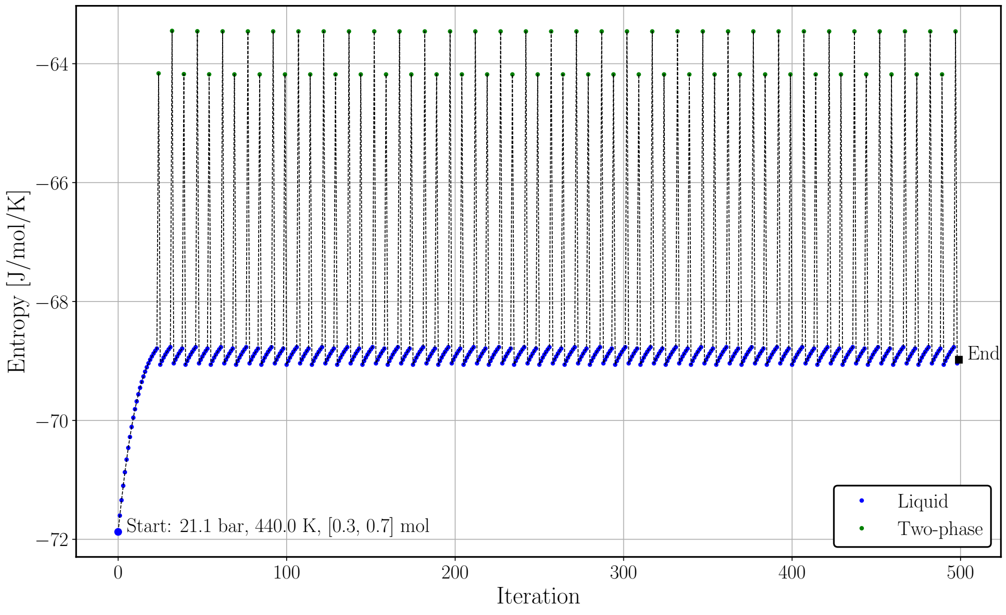


(b)

Figure 5.27: *PT*-*PS*-flash system output temperature (a) and output entropy (b) versus the number of stability test iterations. The system initially seems to quickly start approaching a stable state but decreases somewhat more before eventually reaching a stable value after around 300 iterations.



(a)



(b)

Figure 5.28: *PT*-*PS*-flash system output temperature (a) and output entropy (b) versus the number of stability test iterations. The system reaches a limit cycle after around 25 iteration. Additionally, the system also goes into and out of the two-phase region (or at least is classified as doing so) during the limit cycle.

Vapor region

Table 5.31 shows the main results for the vapor region stability analysis. The table shows that on average the value of temperature after 500 iteration differs around 111 K from the value it started at, for entropy this value is around 26 J/mol/K. Furthermore, for around 35% of cases the phase of the system was different at the end point as compared to the initial state. Similarly to the liquid region the greater majority of cases (around

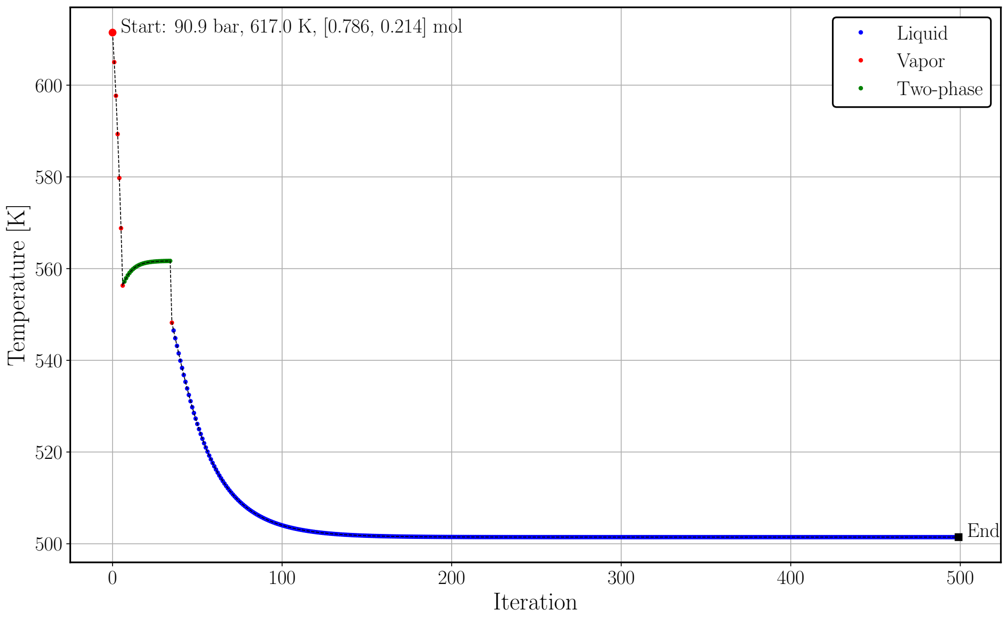
84%) eventually reaches a stable state, no system diverges, and around 16% reach a limit cycle.

Figures 5.29a and 5.29b show the temperature and entropy stability plots for a particularly interesting system that starts in the vapor region, shows a decrease in value for both temperature and entropy, is classified as being in the two-phase region after a few iterations, after which values of temperature and entropy start increasing again; then the system is again classified as vapor for a single iteration and immediately classified as a liquid in the next iteration, which it stays classified as until its values of temperature and entropy slowly reach a stable value after around 200 iterations.

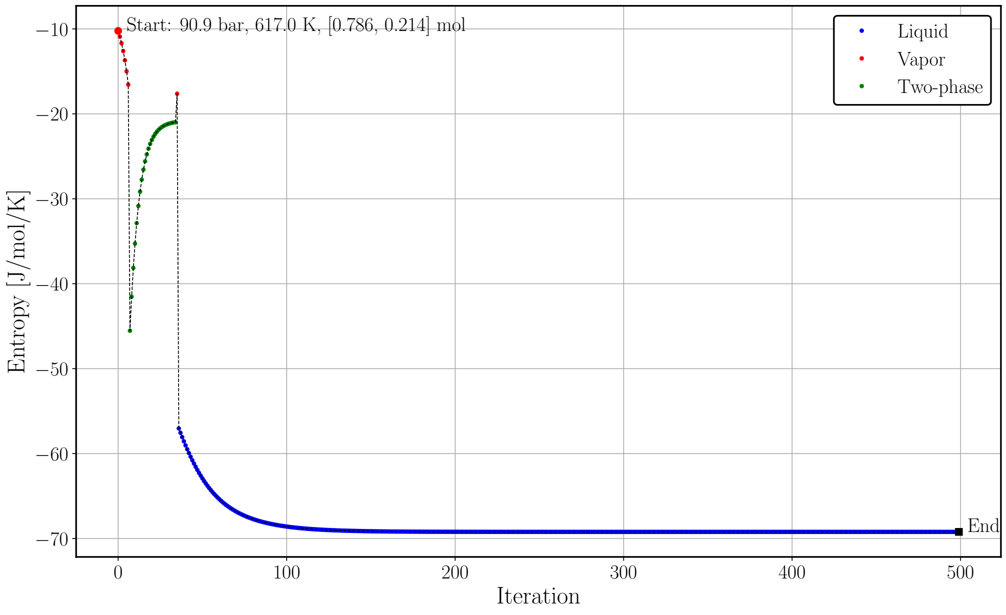
Again, the results show that the system of flashes in question is far from stable.

Table 5.31: Vapor region stability analysis results for the PT-PS-flash system. Mean temperature and entropy deviations indicate the difference between the value of the property at the first iteration and at the last iteration. Percentages of the number properties indicate in how many of the total cases it occurred.

Metric	Value	
Mean temperature deviation	111.3	[K]
Mean entropy deviation	25.9	[J/mol/K]
Number of phase changes	1741	(34.8 %)
Number of converging systems	4175	(83.5 %)
Number of diverging systems	0	(0.0 %)
Number of limit cycle systems	825	(16.5 %)



(a)



(b)

Figure 5.29: *PT-PS*-flash system output temperature (a) and output entropy (b) versus the number of stability test iterations. The system initially shows a rapid decrease in value before entering the two-phase region, in creasing in value seemingly going towards a stable state before again entering (or at least being classified as) the vapor state for one iteration, after which the system is classified as liquid and remains so for the remainder of the iterations after about 200 iterations.

Two-phase region

Table 5.32 shows the main results for the two-phase region stability analysis. After 500 iterations, the average difference between the value of temperature at the initial state and the end state is around 23 K, for entropy this difference is around 26 J/mol/K. In this case almost all (about 94%) of the cases tested eventually ended up in a different phase region from where they started. While a higher percentage was expected when compared to the liquid and vapor regions due to the lower classification accuracy for the two-phase region, a value of 94% is still very high.

Finally, again the majority of cases ends up eventually converging to a stable value though the size of this majority is significantly lower: around 66% for the two-phase region as compared to 88% and 84% for the liquid and vapor regions respectively. Around 34% of the cases ended up reaching a limit cycle.

While the absolute deviations between initial and final state of the values for temperature are much lower compared to the liquid and vapor region, the fact that most systems eventually end up being classified as a different phase than they should be, makes this number somewhat meaningless.

Table 5.32: Two-phase region stability analysis results for the *PT*-*PS*-flash system. Mean temperature and entropy deviations indicate the difference between the value of the property at the first iteration and at the last iteration. Percentages of the number properties indicate in how many of the total cases it occurred.

Metric	Value	
Mean temperature deviation	22.6	[K]
Mean entropy deviation	26.0	[J/mol/K]
Number of phase transition	4693	(93.9 %)
Number of converging systems	3292	(65.8 %)
Number of diverging systems	0	(0.0 %)
Number of limit cycle systems	1708	(34.2 %)

Chapter 6

Conclusions and recommendations

In this work a method was developed for solving vapor-liquid flash problems through the application of artificial neural networks with the aim of decreasing computation times and increasing robustness to failure. The current chapter will provide the main conclusions that were drawn from the results of this work, answer the research questions posed in Chapter 1, as well as provide recommendations for future work.

6.1 Conclusions

In the introduction a number of research questions were posed that guided the development of this project. The next subsections will each repeat and cover one of these research questions. An additional subsection is included which gives some conclusions that were drawn throughout the development of this work, but were not part of the original research questions.

6.1.1 Reframing the flash problem

- ► *In which way can you frame conventional flash problems in order for them to be solvable using artificial neural networks?*

A method was developed that tackles the flash problem in a similar way to conventional flash algorithms: first determining phase stability, then calculating property values using a different model depending on the outcome of the phase stability analysis.

Phase stability was approached as a classification problem, where each phase region (liquid, vapor, two-phase) was given a class label. An artificial neural network was trained to distinguish between phase classes based on given input combinations of pressure-temperature-mole fraction, pressure-entropy-mole fraction, entropy-volume-mole fraction, and enthalpy-volume-mole fraction.

Thermodynamic property prediction was approached as a regression problem using artificial neural networks. Discontinuities in property values that occur naturally due to phase transitions were avoided by training different neural networks on each phase region separately, and using the classification network to decide when to use which network.

6.1.2 New flash types

▶ *Can artificial neural networks be trained to solve one type of flash problem solely based on data obtained from another type of flash problem?*

The simulation engine in which the current artificial neural network fluid flash algorithm was to be used, required two unconventional flash types: an entropy, volume flash, and enthalpy, volume flash. Part of this work included investigating whether these flash types, for which no practically implemented conventional solution framework exists, could be synthesized using artificial neural networks.

In theory, if a relationship exists between two sets of variables, an artificial neural network should be able to be trained that can represent this relationship, provided that enough training data is available. This versatility of representing (non-linear) relations between variables was used to create an SV- and HV-flash using data generated with a PT-flash algorithm only. The data generated using the PT-flash was simply given to the neural network in reverse order, and after it was trained, the network was able to distinguish the reverse relation between them. The new flash types showed similar accuracy (apart from liquid pressures) to the other two more conventional flash types. From this it can be concluded that it is possible to create one type of flash only using data generated using a different type of flash.

In general, the idea of using artificial neural networks to generate a reverse relationship for which no formal mathematical expression or calculation method exists, or for which said method is very complex, is an interesting concept for which many other applications could be found.

6.1.3 Speed improvement

▶ *What speed improvement can be achieved through the use of artificial neural networks?*

In this work speed increases over a conventional method of up to 15 times for most thermodynamic property predictions, and 35 times for phase classification were achieved. The artificial neural networks showed increasing improvements as the number of flash calculations that had to be performed increased. Furthermore, the contribution auxiliary functions had to the total execution time were much higher when the number of flashes that had to be performed was low. From both these results it is concluded that the benefits that can be obtained from neural networks is highest when the number of flashes that have to be executed is large. When only a limited number of flash calculations have to be performed, neural networks can still improve on execution times, but it might not be worth the effort of training them.

6.1.4 Accuracy of the artificial neural networks

▶ *What is the resulting accuracy of the artificial neural networks?*

For the liquid and vapor regions, all classification networks show very high accuracy of over 99.5%. However, points which should lie in the two-phase region are only correctly classified to be so in around 90-93% of the cases, a notably lower number, leading to overall classification accuracy scores of around 97% for all flash types. It is believed that the low two-phase classification accuracy is due to less training data having been available for points in the two-phase region.

The property prediction neural networks show overall decent accuracy scores, with most networks scoring a scaled mean absolute error in the order of 2-9$\times$10^{-3} (on a data set that was scaled to lie roughly linearly within the range [0, 1]), an unscaled MAE that is 0.5%-7% off when compared to the standard deviation of the values within the test data set, and R^2 scores of 0.95 and higher.

The biggest inaccuracies are related to predictions of SV- and HV-flash pressures in the liquid region, which are due to the steep gradients present in the liquid region making it difficult for a neural network to accurately predict property values.

In addition, most predictions in the two-phase region are lower than for the pure liquid and vapor regions, both because of less training data having been available, but mostly due to errors in the phase classification, which lead the neural network flash algorithm to use an incorrect network to predict property values. Furthermore, predictions of properties in the PT two-phase region show additional errors, most likely due to steep gradients which also occur in this phase region.

Finally, current predictions of liquid and vapor phase equilibrium compositions are rather inaccurate. This is partly due to the same reasons as given above, and partly due to the fact that both component vapor fractions from which the phase compositions are derived are independently predicted, leading their errors to build upon each other to create even bigger errors in the values for phase composition.

6.1.5 Increase in robustness

- ► *Can an increase in robustness be achieved through the application of neural networks?*

In general, a neural network will never crash, as it does not include any iteration process it cannot go over a maximum number of iterations or get stuck in a infinite loop. Moreover, when using correct activation functions a neural network will never return a `NaN` or `inf` answer that might result from zero-values being given as inputs to log or exp functions.

Numerical experiments performed show that artificial neural network can still provide reasonably accurate prediction on parts of a data set where data was incorrect or no data was available for training. This is also seen for parts of the classification data set where in certain situations conventional methods incorrectly predicted a certain phase, while the neural network did correctly classify the phase region.

Section 2.3.1 mentioned a couple of situations in which a conventional flash algorithm may fail to provide proper results. This included the algorithm failing due to improper estimates of the initial guess, and failing or slowing down near the critical points, phase boundaries or the STLL. In addition, conventional methods can also provide incorrect results due to a mismatch between the results from phase stability analysis and equilibrium composition calculations.

While the first four situations can be solved using neural networks, as they neither require an initial guess, nor contain any iterative processes that could fail or slow down, the current work has not found a method of preventing the last situation; in fact, the algorithm developed in this work encounters this problem regularly.

6.1.6 Thermodynamic consistency

▶ *To what extent are the resulting neural network flashes thermodynamically consistent?*

The results from the previous chapter show that thermodynamic consistency is hard to achieve through the use of artificial neural networks, a result that was not unexpected. Conventional methods attain consistency by using the same underlying (and intrinsically consistent) equation of state or activation model. For these methods, numerical derivatives can be used to relate one property to another fairly accurately. On the other hand, the accuracy of the neural networks achieved in this work was not high enough for these relation to hold true. Therefore, thermodynamic consistency could not be achieved.

6.1.7 Stability of a system of flashes

▶ *How stable is a system of artificial neural network flash algorithms?*

In this thesis, stability was defined as the tendency of a system to deviate from it's initial state if no changes occur within the system. The results from a number of tests show that a system based on artificial neural network flashes is far from stable, while in most cases a system eventually reaches an constant state, property values at this state differ significantly from their values at the initial state.

This result was also not unexpected as the neural networks used in the current work are in essence black-boxes, with nothing in common apart from the origin of the training data. As a result, their predictions (and therefore inaccuracies) will not be consistent with each other, and the different flashes will therefore struggle to form a consistent and stable interdependent system.

While improving the accuracy of the neural networks will help in decreasing the magnitude of deviation from the initial conditions, it will not make the system truly stable, unless (near-)perfect accuracy can be achieved.

6.1.8 Additional conclusions

The amount of influence that phase classification has on the accuracy of property value predictions was underestimated at the start and during development of this project. Even if incorrect predictions only occur for about 10% of cases, they can still significantly reduce the overall accuracy of subsequent property predictions.

It is believed that the main cause for low classification accuracy in the two-phase region was the fact that less training data was available for points in the two-phase region. While it was known that this was the case, the extent to which this would affect the accuracy of two-phase classification and property prediction accuracy was also severely underestimated.

6.1.9 Main conclusion

The author is of the opinion that neural networks can play an important part in predicting fluid property values in a stand-alone fashion, and believes there is still more room for improvement when it comes to speed and especially accuracy.

However, using a multitude of independent black-box approximators to predict properties that are inherently dependent on one another does not lay a proper foundation for a consistent and stable model of a thermodynamic system.

The current artificial neural network flash algorithm could be used as a fast alternative to get fairly good estimates of many fluid properties, but still requires a significant number of improvements for it to be able to form the basis of a dynamical simulation tool.

6.2 Recommendations

In this section, recommendations that are considered the most important for future work are given. Additional recommendation can be found in Appendix J.

The recommendations considered most vital are:

1. Improving the accuracy of phase classification. It is believed that improving phase classification will also increase property prediction accuracy by preventing the algorithm from using the incorrect neural network. The first step in increasing classification accuracy should be generating training data that is more evenly distributed between the three different phase regions, and generating more data in general. Another interesting method of achieving higher classification accuracy is by generating more data the closer you are to the phase boundaries, and less as you get further away from it, such that the actual phase boundaries are more accurately represented by the data.

2. Developing more advanced data transformations. New methods of scaling and distributing data are required in order to make it possible to properly train neural networks on phase regions in which property values show steep gradients. Important aspects of such a transformation are reversibility (the data set should be able to be transformed back into its original state) and generalizability (the transformation process should be able to be applied to new data points which were not in the original or transformed data set).

3. Finding a better method to predict equilibrium compositions. The current methods used to predict the vapor-liquid equilibrium phase split and phase compositions (x_i and y_i) from component vapor fractions (β_i) are prone to large errors. Due to the fact that the phase compositions are derived from two separate component vapor fractions, errors from those vapor fractions add up in the final values for the phase compositions. A method should be developed that can achieve a higher accuracy, all the while making sure mass balances are not being violated. This could be done by directly predicting the vapor and liquid compositions, but that would not make it possible to also use the results to calculate the total vapor fraction, thus requiring an additional network to be trained (possibly leading to similar problems when it has to be coupled to the phase compositions.

4. Improving thermodynamic consistency and reducing the number of required neural network. This can be done by focussing on increasing the accuracy of fundamental equation predictions, so that other properties can be determined through the use of derivatives of said fundamental equation. If only a single neural network is necessary, more effort can be put into optimizing the topology of the network for accuracy and speed. Alternatively, a hybrid ANN-EOS method similar to [5] could be developed to take advantage of the speed of ANN and the thermodynamic consistency of EOS based methods.

5. Converting the current Python code to C(++) and running it on a GPU. By taking advantage of the efficient manner in which C++ and GPUs can handle large arrays and array computations, a further increase in execution speed can be achieved.

"Give a man a fish, and you feed him for a day; teach a man to fish, and you feed him for a lifetime."

Appendices

Appendix A

The TEA flash algorithm

The flash calculation algorithms used by the COCO simulator [12] and thus the TEA property calculator [11], are based on the algorithms by Boston and Britt [30], and Parekh and Mathias [31].

The Boston and Britt algorithm works by assuming certain simplistic relations between temperature, distribution coefficients and vapor and liquid phase enthalpies, so that the balance equations pertaining to vapor-liquid equilibrium can be rewritten in such a way that a new set of variables emerges that is independent (or negligibly dependent) on the original variables of temperature, vapor fraction, liquid phase mole fractions, and vapor phase mole fractions.

The complete algorithm consists of two loops, one inner-loop and one outer-loop. The inner loop is used to calculate T, x, y, and β which satisfy the enthalpy balance based on a single independent iteration variable. The outer loop assumes a new value for the single independent variable and calculates the values of the coefficients of the assumed temperature correlations until the calculated values match assumed values.

As the new set of variables is independent on temperature, the resulting system of equations is less complex due to less interdependencies between variables being present. Additionally, the inner loop only has to iterate on one variable as opposed to the multiple variables the algorithm from Section 2.2.2 had to iterate on. As a result, the algorithm is very fast to converge.

In order to decrease execution times even more, Boston and Britt also propose using simplistic (polynomial) relations for enthalpy and entropy and other properties in the inner loop, and only using more complex relations (such as an EOS) for more accurate results in the outer loop. In this way, computational load of the inner loop is significantly reduced, and speed of calculation is increased.

The Parekh and Mathias algorithm is an adaptation of the original Boston and Britt algorithm which improves its robustness in extreme situations, and extends the algorithm to additional flash types.

Once an equilibrium composition is found using the Boston-Britt and Parekh-Mathias algorithms, TEA calculates the values of enthalpy and entropy based on an ideal value departure relations and fugacity coefficient derivatives, expressions for which are given in Appendix B. Internal energy, Gibbs free energy, and Helmholtz free energy are calculated using the relations given in Eqs. 2.1. Finally, properties such as heat capacity, viscosity, heat conductivity, and surface tension are calculated using temperature correlations (see Appendix C).

Appendix B

Thermodynamic expressions for selected properties

This appendix gives expressions for a selection of thermodynamic properties. Where relevant, expressions are given using the Peng-Robinson equation of state (EOS) [19].

B.1 Pressure

The Peng-Robinson EOS gives the following explicit expression for pressure as a function of specific volume, temperature and mixture coefficients:

$$P = \frac{RT}{V - b_{\text{mix}}} - \frac{a_{\text{mix}}}{V^2 + 2b_{\text{mix}}V - b_{\text{mix}}^2} \tag{B.1}$$

Here,

P	is the pressure in [Pa],
R	is the universal gas constant (8.134 J/mol/K),
T	is the temperature in [K],
V	is the volume in [m^3/mol],
a_{mix}	is the mixture attraction parameter in [Jm^3/mol^2], and
b_{mix}	is the mixture intrinsic volume parameter in [m^3/mol].

The values of a_{mix} and b_{mix} are often calculated using the following mixture rules [104]:

$$a_{\text{mix}} = \sum_{i=1}^{c}\sum_{j=1}^{c} z_i z_j a_{ij} \tag{B.2a}$$

$$b_{\text{mix}} = \sum_{i=1}^{c} z_i b_i \tag{B.2b}$$

$$a_{ij} = (1 - k_{ij})\sqrt{a_i a_j} \tag{B.2c}$$

And pure component expressions [19]:

$$b_i = 0.078\frac{RT_{\text{c},i}}{P_{\text{c},i}} \tag{B.3a}$$

$$a_i = 0.45724\frac{R^2T_{\mathrm{c},i}^2}{P_{\mathrm{c},i}}[1 + m_i(1 - \sqrt{T_{\mathrm{r},i}})]^2 \tag{B.3b}$$

$$T_{\mathrm{r},i} = \frac{T}{T_{\mathrm{c},i}} \tag{B.3c}$$

$$m_i = \begin{cases} 0.37464 + 1.54226\omega_i - 0.26992\omega_i^2, & \text{if } \omega_i < 0.5 \\ 0.3796 + 1.485\omega_i - 0.1644\omega_i^2, & \text{if } \omega_i \geq 0.5 \end{cases} \tag{B.3d}$$

Here,

c	is the total number of components present in the mixture,
z_i	is the mole fraction of components i in [mol/mol],
k_{ij}	is the binary interaction parameter between components i and j
a_i	is the attraction parameter of component i in [Jm^3/mol^2],
b_i	is the intrinsic volume parameter of component i in [m^3/mol],
R	is the universal gas constant (8.134 J/mol/K),
$T_{\mathrm{c},i}$	is the critical temperature of component i in [K],
$P_{\mathrm{c},i}$	is the critical pressure of component i in [Pa],
$T_{\mathrm{r},i}$	is the reduced temperature of component i, and
ω_i	is the acentric factor of component i.

B.2 Compressibility factor

The compressibility factor indicates the deviation of a given thermodynamic system from the ideal gas law, and is defined as:

$$Z = \frac{PV}{RT} \tag{B.4}$$

It is an integral part of determining the fugacity of a system. When a system is in the two-phase region, both phases have a different compressibility factor.

For a cubic equation of state, the compressibility factor is found at a known pressure by finding the roots of the equation of state at that pressure. Mathematically speaking, this is done for the Peng-Robinson EOS by solving the following equation for Z:

$$Z^3 - (1-\mathcal{B})Z^2(\mathcal{A} - 2\mathcal{B} - 3\mathcal{B}^2)Z - (\mathcal{A}\mathcal{B} - \mathcal{B}^2 - \mathcal{B}^3) = 0 \tag{B.5}$$

Here,

$$\mathcal{A} = \frac{a_{\mathrm{mix}}P}{R^2T^2} \tag{B.6a}$$

$$\mathcal{B} = \frac{b_{\mathrm{mix}}P}{RT} \tag{B.6b}$$

As the above expression for the compressibility factor is cubic, it has either one or three *distinct* and *real* roots. Whenever a system has only one distinct real root, it means it is either supercritical, in which case the other two roots are complex values (which have no physical meaning here), or the system is exactly critical, in which case it has three identical roots (so only one distinct root). If a system is subcritical it will have three distinct real roots, in this case the root with the lowest value corresponds to the liquid phase (and can be used to calculate the liquid specific volume), and the root with the highest value corresponds to the vapor phase (and can be used to calculate the vapor specific volume). The root with the middle value has no physical meaning and is not used.

B.3 Fugacity

The fugacity of a real fluid is an expression for its effective partial pressure. Every component in a mixture has its own corresponding fugacity. In addition, for a two-phase system, every component has two fugacity values, one for the liquid phase, and one for the vapor phase. At equilibrium, both fugacity values are equal.

The following expression is used for the Peng-Robinson EOS to calculate fugacity of component i:

$$f_i = P z_i \exp\left[BB_i(Z-1) - \log(Z-\mathcal{B}) - \mathcal{I}\right] \tag{B.7a}$$

$$\mathcal{I} = \frac{\mathcal{A}}{2\sqrt{2}\mathcal{B}}(AA_i - BB_i)\log\left(\frac{Z+\mathcal{B}(1+\sqrt{2})}{Z+\mathcal{B}(1-\sqrt{2}}\right) \tag{B.7b}$$

Here, P is the pressure (in Pa) in the system, Z the compressibility factor, and the remaining factors are calculates as follows:

$$BB_i = \frac{b_i}{b_{\text{mix}}} \tag{B.8a}$$

$$AA_i = \frac{2}{a_{\text{mix}}}\sum_{j=1}^{c} z_i a_{ij} \tag{B.8b}$$

The fugacities of either the liquid and vapor phase can be determined by plugging in the compressibility factor Z corresponding to each phase respectively.

B.4 Enthalpy

Enthalpy is often calculated based on a departure function. The TEA calculator [11] uses a departure from the ideal gas value:

$$H = H_{\text{id}}^{\text{V}} - RT^2\sum_{i=1}^{c} z_i\left(\frac{\partial\ln(\phi_i)}{\partial T}\right)_P \tag{B.9a}$$

$$H_{\text{id}}^{\text{V}} = \sum_{i=1}^{c} z_i \int_{T_{\text{ref}}}^{T} C_{p,i}\mathrm{d}T \tag{B.9b}$$

Here,

H	is the enthalpy in [J/mol],
H_{id}	is the ideal gas enthalpy in [J/mol],
ϕ_i	is the fugacity coefficient ($\phi_i = \dfrac{f_i}{P_i z_i}$),
T_{ref}	is the temperature at a reference state in [K], and
$C_{p,i}$	is the heat capacity of component i in [J/mol/K].

B.5 Entropy

Entropy is calculated through a similar departure from its ideal gas value:

$$S = S_{\text{id}}^{\text{V}} - \left[\sum_{i=1}^{c} z_i R \ln(\phi_i) + RT\left(\frac{\partial \ln(\phi_i)}{\partial T}\right)_p\right] \tag{B.10a}$$

$$S_{\text{id}}^{\text{V}} = \sum_{i=1}^{c} z_i \left[-R\ln\left(\frac{P}{P_{\text{ref}}}\right) + \int_{T_{\text{ref}}}^{T} \frac{C_{p,i}}{T}\mathrm{d}T - R\ln(z_i)\right] \tag{B.10b}$$

Here,

S	is the entropy in [J/mol/K],
H_{id}	is the ideal gas entropy in [J/mol/K], and
P_{ref}	is the pressure at a reference state in [Pa].

Appendix C

Temperature correlations

In this appendix the temperature correlations used by the TEA property calculator [11] and the ANN property calculator are given. All correlations and were taken from [105] and its related software package.

C.1 Heat Capacity

Heat capacity is calculated based on a experimental temperature correlations. Heat capacity for the vapor phase are considered to be equal to the ideal gas values, and thus use the ideal gas correlation. The liquid phase heat capacity used the same correlation equation form, but with different constants.

$$C_{p,i} = A_i + \exp\left(\frac{B_i}{T} + C_i + D_i T + E_i T^2\right) \tag{C.1}$$

Here, T is the temperature in Kelvin, and A_i, B_i, C_i, D_i, and E_i are experimentally determined pure compound specific constants (which will not be given here). Based on the pure compound heat capacities of the individual components, the total mixture heat capacity (in J/mol/K) is calculated by:

$$C_p = \frac{\sum_{i=1}^{c} C_{p,i} z_i}{1000} \tag{C.2}$$

C.2 Viscosity

The viscosity of a pure compound (in Pa·s) is calculated based on a experimental temperature correlation. The correlation differs for the liquid and vapor phase, and are given by:

$$\eta_i^{\mathrm{L}} = A_i + \frac{B_i}{T} + C_i \log T + D_i T^{E_i} \tag{C.3a}$$

$$\eta_i^{\mathrm{V}} = \frac{A_i + T^{B_i}}{1 + \frac{C_i}{T} + \frac{D_i}{T^2}} \tag{C.3b}$$

Here, T is the temperature in Kelvin, and A_i, B_i, C_i, D_i, and E_i are experimentally determined pure compound specific constants. The values of the constants differ for both correlations, and will not be given here. The total viscosity of each phase in a given mixture in is calculated as:

$$\eta^{\mathrm{L}} = \sum_{i=1}^{c} \eta_i x_i \tag{C.4a}$$

$$\eta^{\mathrm{V}} = \sum_{i=1}^{c} \eta_i y_i \tag{C.4b}$$

Here, x_i is the liquid composition, and y_i is the vapor composition.

C.3 Thermal Conductivity

As for viscosity, the values of the liquid and vapor thermal conductivities (in W/m/K) are given by experimental temperature correlations:

$$\lambda_i^{\mathrm{L}} = A_i + \exp\left(\frac{B_i}{T} + C_i + D_i T + E_i T^2\right) \tag{C.5a}$$

$$\lambda_i^{\mathrm{V}} = \frac{A_i + T^{B_i}}{1 + \frac{C_i}{T} + \frac{D_i}{T^2}} \tag{C.5b}$$

The total thermal conductivity of each phase in a given mixture in is calculated as:

$$\lambda^{\mathrm{L}} = \sum_{i=1}^{c} \lambda_i x_i \tag{C.6a}$$

$$\lambda^{\mathrm{V}} = \sum_{i=1}^{c} \lambda_i y_i \tag{C.6b}$$

Here, x_i is the liquid composition, and y_i is the vapor composition.

Appendix D

Additional optimization concepts

D.1 Local versus global optimization

Most non-linear optimization problems, especially multivariate ones, do not have a single minimum or maximum but many, all at different locations in their input domain. However, out of all the minima, there is generally only one which attains the absolute lowest value of the objective function, and vice versa for the maxima. The optimum which attains the absolute highest or lowest value of all possible values in the objective functions range is called the *global* optimum, all other optima are called *local* optima [40, 67, 68]. The goal of optimization is thus finding the global optimum, while avoiding converging to local optima.

Unfortunately, finding an objective functions global optimum is much more complex than finding a local optimum, requiring more time and resources [40]. The gradient descent algorithm described in Section 3.4.2 has (most of the time) not much trouble in finding a given objective functions optimum, however, this optimum cannot be guaranteed to be the global optimum, and could just as or more likely be a local optimum.

For many algorithms the reason why the algorithm converges to a local optimum is due to its starting point, or initial guess. If the initial guess is not within the *region of attraction* (all the points in the search-space that lead to a (local) optimum if the slope of the function is followed upwards or downwards) of the global optimum, the optimization algorithm will never converge to the global optimum as it will find a different local optimum before it reaches the global optimums region of attraction [40, 67]. The difference between local and global convergence is shown in Figure D.1 for a one-dimensional problem.

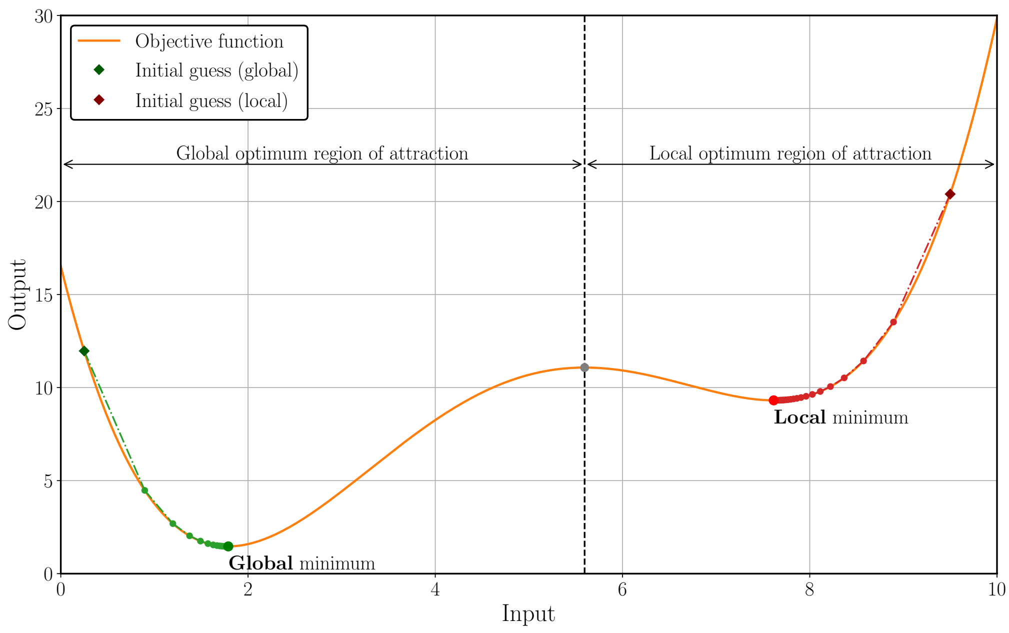


Figure D.1: Example of global versus local convergence of the gradient descent algorithm. If the starting point lies outside of the global optimums region of attraction, the algorithm will find a local minimum instead.

The problem of an inaccurate initial guess leading to local instead of global convergence could potentially be solved by starting with many different initial guesses, finding the nearest optimum to each guess, and choosing the optimum with the best result as the global optimum [68]. In fact, many global optimization algorithms function by solving multiple local optimization problems [40].

Other methods of global optimization include stochastic algorithms, which apply a certain degree of randomness to their iteration steps in order to explore a larger region of the search-space, and avoid converging to local solutions [106, 107], or use more complicated math, such as interval arithmetic [108]. However, for some problems global convergence can only be guaranteed if the entire search-space is explored, as otherwise there might always be part of the search-space that was not explored but does contain the global optimum [68]. Nonetheless, many global optimization algorithms generally do a very good job of finding a satisfactory global solution to a given problem.

The biggest downside to global optimization is that most algorithm require a long time to converge to a satisfactory solution, so for certain time-constrained problems, local optimization with good initial guess estimation is preferred.

D.2 Deterministic versus stochastic optimization

The optimization algorithm described in Section 3.4.2 is an example of a *deterministic* algorithm: if all settings (initial guess, step size α, convergence criterion, maximum number of iterations, etc.) are kept constant, the algorithm will always return the exact same results.

On the other hand, a *stochastic* algorithm incorporates some form of randomness in order to find the optimum of a given problem. As a result, a stochastic algorithm will not return the same results for subsequent executions of its code. For example, while a deterministic algorithm will always return a value of 1.9982 on consecutive executions for constant settings, a stochastic algorithm will return 1.9972 the first time, 2.0021 the next, 1.9905 for another execution, 1.999 for again another, etc., all for the exact same settings.

Stochastic methods are often employed in global optimization algorithms to efficiently explore the problems search-space and avoid the algorithm from getting stuck in local optima. While a stochastic algorithm is not guaranteed to find a global optimum, it has a bigger chance of finding it than a simple deterministic gradient-based algorithm, and is much less dependent on the value of the initial guess. Oftentimes, stochastic algorithms are executed a number of times in order to verify whether the algorithm converges to the same result for the majority of executions, e.g. if you run an algorithm 50 times, and it converges to the same answer 43 times, you can be fairly certain that answer is the global optimum.

An additional reason for using stochastic optimization algorithms is when the objective function considered is non-smooth or even discontinuous. In this case the gradient of the objective function is either discontinuous or non-existent for parts of its range, leading to the breakdown of general gradient-based optimization algorithms, and stochastic algorithms will result in much more accurate results [109].

Appendix E

Algorithm design alternatives

In this appendix two algorithm design alternatives are detailed which were considered at early design stages, but ultimately deemed either to complicated, slow, inaccurate, or convoluted.

E.1 All-in-one approach

E.1.1 Description

One of the most straightforward neural network flash architectures would be to include the full range of inputs and outputs in one network, i.e. create a single large network that can be given any of the input combinations to predict any of the possible properties. The major benefit of using a single network for all calculations is the simplicity of the complete algorithm, there is very little need for additional calculations or processing methods making the entire pipeline from input to output very streamlined. However, while this approach would be one of the simplest architectures in terms of ease of algorithm design, there are many difficulties when it comes to training one network on so many variables.

The first issue comes from an increase in dimensionality of the weight optimization problem. As discussed previously, training of a neural network is primarily an optimization problem, in which the goal is to find the optimal weights (and biases) which lead to the minimum value of the specified loss function. In general, the dimensionality of an optimization problem (the number of variables that influence the value of the objective function) has a big influence on how quickly it finds a solution and whether or not that solution is a good solution [40]. When the number of inputs and outputs increases, the network size (number of weights and biases) increases as well, and as a result execution time will increase.

While the previous problem can make training more time consuming and difficult, it is not insurmountable. A much more difficult problem comes from training one network on the entire range of inputs. In Chapter 2 it was mentioned that any mixture of chemical compounds can exhibit different phase regions, depending on its exact composition and the values of two thermodynamic variables. Which phase region a given mixture is in can have a large effect on the values of the remaining thermodynamic properties of the mixture. For instance, the density of water changes from around 1000 kg/m^3 in the liquid state to around 1 kg/m^3 in the vapor state. In addition, the boundary between phases is usually rather abrupt, meaning a small change in one thermodynamic property can result

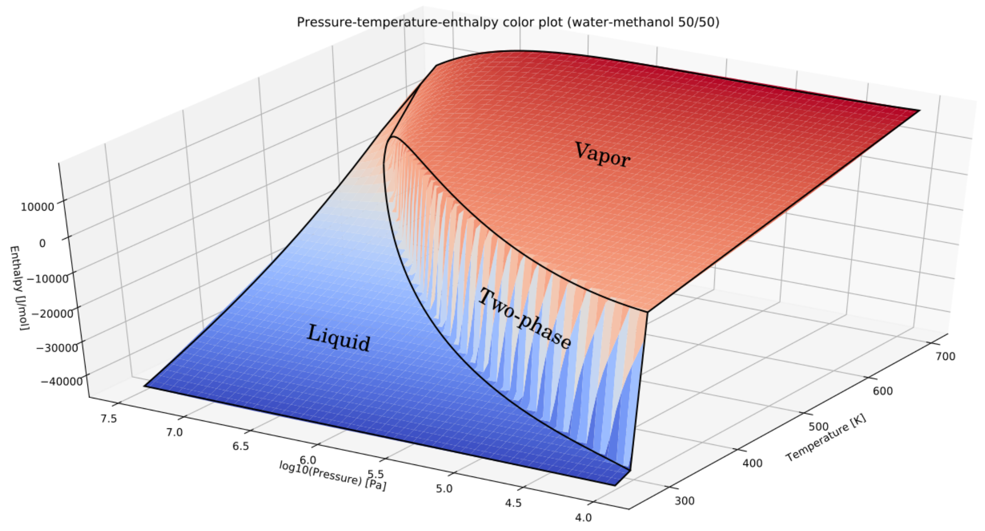


Figure E.1: Pressure-temperature-enthalpy surface plot with phase region indication.

in large changes to other properties when crossing a phase boundary, see for example Figure E.1 which shows the enthalpy as a function of pressure and temperature for an equimolar mixture of water and methanol. As can be seen from the figure, for pressures below the critical point, enthalpy increases vary rapidly as a function of temperature during the transition from the liquid region through the two-phase region to the vapor region.

It is exactly these types of discontinuities that make training a neural network on all phase regions at once difficult. As was stated in Chapter 3, a neural network with a single hidden layer can approximate any function to any degree of accuracy, however, the function must be *continuous*, any discontinuity in the function will inevitably lead to inaccuracies and oscillations around the point(s) of discontinuity. This is a problem that is not easily solved by increasing training duration, increasing the number of layers of the network, or adding more training data.

A very basic accuracy comparison was made between this approach and the chosen design which showed that for enthalpy the MAE of the all-in-one approach was around 260 J/mol, while for the chosen approach it was around 175 J/mol, which further discouraged pursuing the all-in-one approach.

One final disadvantage of this method is that for most calculations, a large parts of the network will be redundant, i.e. a network will always calculate all outputs, even when only one is required by the user. Since a neural network is a black box it cannot be determined exactly which weights and biases influence which outputs, and thus it is impossible for a neural network to run through only a part of it's weights and biases so only one output is calculated, it is either all outputs or none. Therefore, using a single network to link all inputs and outputs will lead to a rather computationally inefficient network in the final application.

E.1.2 Not chosen because

Discontinuities make it difficult to accurately predict the boundaries between phase regions. Preliminary accuracy comparisons indicated other methods could achieve higher accuracy. Predicting many properties with a single network can lead to the network becoming complex and slow, and training can take very long. Additionally, most of the network would be redundant if only one property is required.

E.2 The golden mean approach

E.2.1 Description

One way to avoid the discontinuity problems discussed in the previous section is to split the entire input range based on the three phase regions (liquid, vapor, two-phase) and training one neural network on each phase. As all thermodynamic variables and properties within the same phase region are generally smooth and continuous, any discontinuity can be avoided in this way.

The main challenge with this approach is determining which of the networks should be used for which inputs. Most of the time, when a certain input is given (for example pressure, temperature, and composition), it is not directly known to which phase region these inputs belong, and thus which of the three networks should be used to make the correct prediction.

One way of potentially solving this issue is as follows: for every phase region a network is trained over the entire input domain. For values of the input range where a certain phase region does not exist, data is extended (by adding dummy values) in a way that assures continuity and smoothness of the resulting curve (to avoid discontinuities). In addition to continuity and smoothness, the data is extended in such a way that the for each of the phase regions, the neural network corresponding to said phase region predicts a value that is in the middle of all three network prediction values (i.e. the curve of the correct phase is in between the other two curves).

Figure E.2 gives an example of this method. At small volumes, only the liquid phase region is present (solid red line), at increased volumes only the two-phase region is present (solid blue line), and at even higher volumes only the vapor phase region exists (solid green line). Pseudo-data is created in order to smoothly and continuously extend each phase curve into the phase regions where it isn't normally present (dotted lines). By correctly generating the pseudo-data, it can be insured that every curve is smooth, and the solid phase lines will always be in between the dotted lines in their respective phase regions, i.e. the solid red line lies in between the other lines only in the liquid phase region, the solid blue line is in the middle only in the two-phase region, etc. The pseudo-data has no physical meaning and is purely used as a tool.

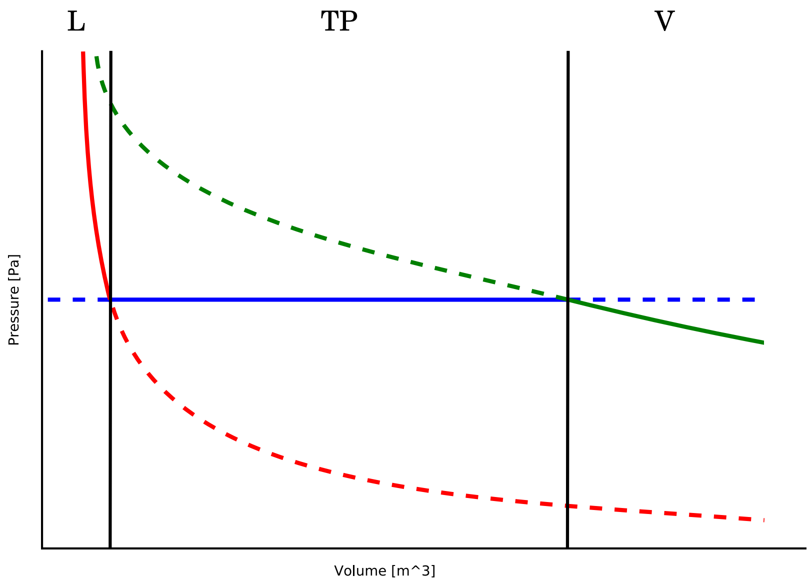


Figure E.2: Example of how data can be extended in such a way that the resulting curves are continuous and smooth, and that each phase curve in in between the others in its corresponding phase region. Where the red line indicates the liquid phase (L), the blue line the two-phase region (TP), and the green line vapor phase (V). The solid parts of the line indicate the parts of the curve, and the dashed parts indicate the extended parts.

As the extended data curves are all smooth and continuous, a neural network should be able to be trained on it to achieve high accuracy. Once the networks are properly trained, predictions can be made by giving all networks the same input and collecting their predictions, the network whose predicted value lies in between the other two predictions is taken as the true prediction.

One issue with this approach is the way in which the data should be extended in order to insure the right curve is always in the middle for each phase region. Of course, the extended liquid and vapor curves need to approach the same asymptote, but if they approach this asymptote at the wrong rates, one might eventually cross the other, which would lead to incorrect predictions, as is illustrated in Figure E.3.

Another, bigger, problem comes with the increase of the number of dimensions of the problem. The data in Figure E.2 seems straight-forward to extend, but this figure only represents a single composition. However, as the number of components increases, the number of different compositions increases exponentially, as does the number of curves such as shown in Figure E.2. As some basic information on the shape of the curve of each composition is required in order to extend the data correctly, a huge amount of data is already needed just to be able to extend the data, add to this the extended data, plus the original data itself, and the size of your data sets rapidly becomes unmanageable.

Another downside to this method is that at least three networks need be called upon to make only a single prediction, making this method somewhat computationally inefficient.

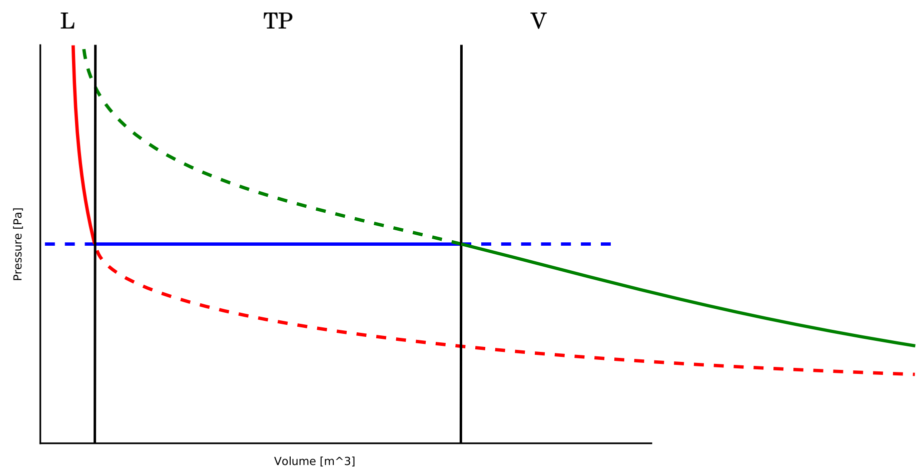


Figure E.3: Example of incorrectly extended data. In this example both the solid green line, and the dashed red line asymptotically approach zero, but not at the same rates. The dashed red line will eventually cross the solid green line, leading to incorrect predictions as the correct line is no longer the middle of the three.

E.2.2 Not chosen because

Data extension will most likely become a complex and convoluted process that requires knowledge of the shape and limits of the data beforehand. The amount of data required is also large as pseudo-data has to created in places where otherwise no data exists. Especially if the number of dimensions increases (more components are added), the amounts of data will likely become extremely large.

Appendix F

Phase composition error propagation

In this appendix it will be proven that the absolute error scores for the vapor phase mole fractions (y_w and y_m) are always equal for both components, even if the relative error of the predictions for the component vapor fractions (β_w and β_m) differ. It will also be shown that a percentage based score will not be equal.

The vapor-liquid equilibrium phase compositions y_i and x_i are in this thesis calculated based on predictions of the component vapor fractions β_i defined as:

$$\beta_i = \frac{n_i^\mathrm{V}}{n_i^\mathrm{F}} \tag{F.1}$$

And can be used to calculate phase compositions as follows:

$$n_i^\mathrm{V} = \beta_i n_i^\mathrm{F} \tag{F.2a}$$

$$n_i^\mathrm{L} = n_i^\mathrm{F} - n_i^\mathrm{V} \tag{F.2b}$$

$$y_i = \frac{n_i^\mathrm{V}}{\sum_{i=1}^{c} n_i^\mathrm{V}} \tag{F.2c}$$

$$x_i = \frac{n_i^\mathrm{L}}{\sum_{i=1}^{c} n_i^\mathrm{L}} \tag{F.2d}$$

Here,

β_i	is the vapor fraction of component i in [mol/mol],
n_i	is the number of moles of component i present in [mol],
x_i	is the mole fraction of component i in the liquid phase in [mol/mol],
y_i	is the mole fraction of component i in the vapor phase in [mol/mol],
c	is the total number of components in the mixture, and

V, L, and F are superscripts indicating the vapor phase, liquid phase, and original feed respectively.

Substituting for the components of water and methanol using the subscripts w and m, the expressions for the vapor phase mole fractions become:

$$y_{\mathrm{w}} = \frac{\beta_{\mathrm{w}} n_{\mathrm{w}}^{\mathrm{F}}}{\beta_{\mathrm{w}} n_{\mathrm{w}}^{\mathrm{F}} + \beta_{\mathrm{m}} n_{\mathrm{m}}^{\mathrm{F}}} \tag{F.3}$$

$$y_{\mathrm{m}} = \frac{\beta_{\mathrm{m}} n_{\mathrm{m}}^{\mathrm{F}}}{\beta_{\mathrm{w}} n_{\mathrm{w}}^{\mathrm{F}} + \beta_{\mathrm{m}} n_{\mathrm{m}}^{\mathrm{F}}} \tag{F.4}$$

Introducing the component vapor fraction prediction errors ϵ_{w} and ϵ_{m}:

$$\beta_{\mathrm{w}} = \beta_{\mathrm{w}} + \epsilon_{\mathrm{w}} \tag{F.5}$$

$$\beta_{\mathrm{m}} = \beta_{\mathrm{m}} + \epsilon_{\mathrm{m}} \tag{F.6}$$

The expressions for the vapor mole fractions become:

$$y'_{\mathrm{w}} = \frac{\beta_{\mathrm{w}} n_{\mathrm{w}}^{\mathrm{F}} + \epsilon_{\mathrm{w}} n_{\mathrm{w}}^{\mathrm{F}}}{\beta_{\mathrm{w}} n_{\mathrm{w}}^{\mathrm{F}} + \beta_{\mathrm{m}} n_{\mathrm{m}}^{\mathrm{F}} + \epsilon_{\mathrm{w}} n_{\mathrm{w}}^{\mathrm{F}} + \epsilon_{\mathrm{m}} n_{\mathrm{m}}^{\mathrm{F}}} \tag{F.7}$$

$$y'_{\mathrm{w}} = \frac{\beta_{\mathrm{m}} n_{\mathrm{m}}^{\mathrm{F}} + \epsilon_{\mathrm{m}} n_{\mathrm{m}}^{\mathrm{F}}}{\beta_{\mathrm{w}} n_{\mathrm{w}}^{\mathrm{F}} + \beta_{\mathrm{m}} n_{\mathrm{m}}^{\mathrm{F}} + \epsilon_{\mathrm{w}} n_{\mathrm{w}}^{\mathrm{F}} + \epsilon_{\mathrm{m}} n_{\mathrm{m}}^{\mathrm{F}}} \tag{F.8}$$

Substituting A for $\beta_{\mathrm{w}} n_{\mathrm{w}}^{\mathrm{F}}$, B for $\beta_{\mathrm{m}} n_{\mathrm{m}}^{\mathrm{F}}$, γ for $\epsilon_{\mathrm{w}} n_{\mathrm{w}}^{\mathrm{F}}$ and δ for $\epsilon_{\mathrm{m}} n_{\mathrm{m}}^{\mathrm{F}}$, the following expressions are obtained which are somewhat more easy on the eyes:

$$y'_{\mathrm{w}} = \frac{A + \gamma}{A + B + \gamma + \delta} \tag{F.9}$$

$$y'_{\mathrm{m}} = \frac{B + \delta}{A + B + \gamma + \delta} \tag{F.10}$$

In order to determine the absolute error, the expressions for y'_{w} and y'_{m} are subtracted from the original (error-less) expressions for y_{w} and y_{m}:

$$y_{\mathrm{w}} - y'_{\mathrm{w}} = \frac{A}{A + B} - \frac{A + \gamma}{A + B + \gamma + \delta} \tag{F.11}$$

$$y_{\mathrm{m}} - y'_{\mathrm{m}} = \frac{B}{A + B} - \frac{B + \delta}{A + B + \gamma + \delta} \tag{F.12}$$

Finally, by combining the fractions in the above equations and rewriting we get the expressions:

$$y_{\mathrm{w}} - y'_{\mathrm{w}} = \frac{A\delta + B\gamma}{A^2 + B^2 + 2AB + A\gamma + A\delta + B\gamma + B\delta} \tag{F.13}$$

$$y_{\mathrm{m}} - y'_{\mathrm{m}} = \frac{A\delta + B\gamma}{A^2 + B^2 + 2AB + A\gamma + A\delta + B\gamma + B\delta} \tag{F.14}$$

Which are equal, thereby showing that the absolute error score for the mole fraction of water will always be equal to that of methanol. Furthermore, it also shows that both absolute errors are equal, even if one the β_i predictions has a much higher error score, and that the errors from both predictions add up in the final value of the vapor phase mole fractions.

Dividing the above expressions again by the expressions for y_{w} and y_{m} respectively, the following expressions for the fractional (percentage) error are obtained:

$$\frac{y_{\mathrm{w}} - y'_{\mathrm{w}}}{y_{\mathrm{w}}} = \frac{A\delta + B\gamma}{A^2 + B^2 + 2AB + A\gamma + A\delta + B\gamma + B\delta} \frac{A + B}{A} \tag{F.15}$$

$$\frac{y_{\mathrm{m}} - y'_{\mathrm{m}}}{y_{\mathrm{m}}} = \frac{A\delta + B\gamma}{A^2 + B^2 + 2AB + A\gamma + A\delta + B\gamma + B\delta} \frac{A + B}{B} \tag{F.16}$$

Which shows that unless A and B are the same, one of the expression will always evaluate to a higher number than the other. Since A and B correspond to the values of vapor mole numbers of water and methanol, this will only occur if the vapor phase mole fractions are equal. This approach can also be used to prove the same facts for the liquid phase mole fractions.

Appendix G

Critical point predictions based on incomplete data sets

In order to show that neural networks can still provide decent results when the data they have been trained on were partially incomplete, a numerical experiment has been performed. The experiment focusses on predicting values of the critical point of pure components using neural networks trained on a data set missing points near the critical region.

G.1 Critical prediction: methods

As this investigation looks at the amount of data necessary to still make accurate predictions around the critical point, it is important to have property values over the entire relevant input range, including the critical point. It was therefore decided to start off with pure compound data of two well-studied fluids: carbon dioxide (CO_2) and methane (CH_4).

In order to investigate the minimum temperature up to which data would be required to be able to make fairly accurate predictions of the critical point, multiple data sets were prepared each containing vapor-liquid coexistence densities up to a temperature whose value was a specified fraction of the critical temperature.

Data was generated using REFPROP starting from temperatures well below the critical temperature of each compound. In both cases exactly 2000 data points were gathered, of which 1400 were to be used as training data for the artificial neural network (ANN).

After each network was trained to a sufficiently high accuracy on the test set, their accuracy score on the control curve was determined, as well as their accuracy in predicting critical temperature and density. The critical temperature and density of each network were determined by predicting temperatures over a wide range of finely spaced densities; the density at which the predicted temperature showed a maximum was considered the predicted critical density and the corresponding temperature the predicted critical temperature.

Results from the neural network predictions were compared to conventional methods for predicting critical values of a fluid, such as the scaling law of rectilinear diameter [110].

G.1.1 Number of neurons in the hidden layer

As an initial proof-of-concept, this study uses a straightforward Multi-layer Perceptron (MLP) neural network with a single hidden layer. The MLP is one of the most basic type

Table G.1: Overview of settings used in constructing and training the artificial neural networks.

Setting	Value
Hidden layers	1
Hidden neurons	3 or 4
Hidden layer activation function	tanh
Output layer activation function	linear
Train-test-validation split [%]	70-15-15
Training optimization algorithm	Levenberg-Marquardt

of artificial neural networks, but is very versatile in its application and easy to use. The parameter which has the biggest influence on the final accuracy of the trained network is the number of neurons in the hidden layer. In general, more neurons will lead to a higher accuracy on the training data, but is less generalizable outside the range of the training data, on the other hand, a network with less neurons generally has a slightly lower accuracy on the training data, but will be more generalizable, this is shown in Figure G.1.

During this study it was found that networks with 3 or 4 neurons in the hidden layer were best suited to solve the problem at hand. Any less than three and the resulting fit would not be accurate enough, and any more than 4 would lead to non-generalizable fits such as the 10 neuron fit of Figure G.1. As both 3 and 4 hidden neuron networks would often lead to fits with different shapes, the remainder of this study will compare the scores of both 3 neuron and 4 neuron networks on the different performance metrics.

G.1.2 Neural networks parameters

Apart from the difference in number of neurons in the networks hidden layer, all other parameters were kept exactly the same for each artificial neural network that was to be trained. These parameters and settings include: number of total data points, percentage of data points used as training data, test data, and validation data, hidden and output layer activation function, and the training algorithm. Table G.1 gives an overview of the settings used to train each network in this study.

G.1.3 Performance metrics

The neural networks performances on how well they can predict the values of a fluids critical temperature, density, and pressure were scored based on a absolute percentage error, calculated as follows:

$$\text{APE} = \frac{|t - o|}{t} \cdot 100 \tag{G.1}$$

Here,

APE	is the absolute percentage error,
t	is the target value, and
o	is the predicted value.

Predictions made using the law of rectilinear diameter were scored in exactly the same way. The APE was used despite its downsides as the values predicted here all have an absolute zero point, and no severely skewed results are obtained as no mean values need to be determined.

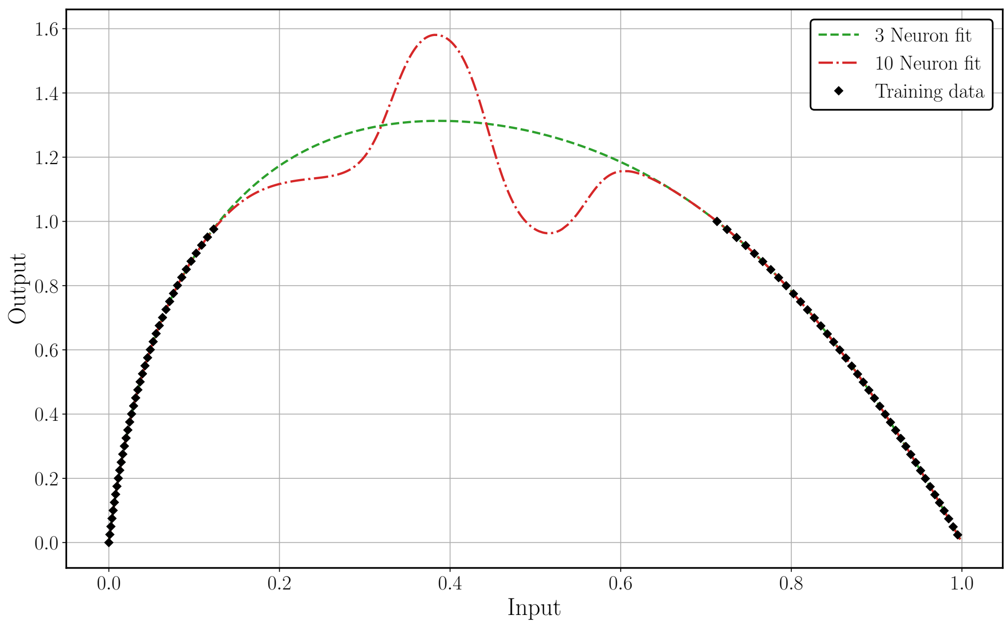


Figure G.1: Comparison between two artificial neural network fits on a data sets which includes a large gap in data. Both were trained with exactly the same settings apart from the number of neurons in the hidden layer of the network. As can be seen, both networks give accurate fits of the training data, but the 10 neuron network shows very erratic behavior in the range of the data gap, while the 3 neuron network is much more well-behaved.

G.2 Critical point predictions: results

G.2.1 Critical temperature

Table G.2 and G.3 show the accuracy of the critical temperature predictions for methane and carbon dioxide respectively. The tables compare results obtained using the neural network with three hidden neurons (3H), four hidden neurons (4H), and finally the scaling law of rectilinear diameter (SLDR).

As can be seen from Table G.2, for methane the network with only three hidden neurons generally outperforms the network with four hidden neurons except for the factor = 0.6 and factor = 1.0 cases. The result that the 3H network struggles with the entire data set (factor = 1) has been found in many cases, even ones not shown here. This phenomenon is most likely caused by the fact that the three hidden neurons are to restrictive for the network to fit to the complete data set, but not when it only has to fit a part of it. This conclusion is supported by the fact that the 4H network (which is less restricted) has no problem accurately fitting the entire data set.

On the other hand, the fact that the 4H network is less restricted also leads it to less accurately fit when less data is available. As can be seen from Figure G.3, the 4H network flattens out where no data is available, while Figure G.2 shows that the 3H network keeps a rounded curve in the part where no data was available for training.

Table G.2 also seems to indicate that (at least for the 3H network) higher accuracy is achieved for lower factors, which goes against what one might expect. This could be due to the fact that the neural networks were trained using a stochastic algorithm, and thus the outcome is somewhat random. As a result, it can happen that a network trained on less complete data "accidentally" achieves a higher accuracy than a network trained more complete data. In order to mitigate this stochastic effect, each network should have been trained multiple times, after which the average accuracy from all networks would should have been determined to get a better indication of accuracy. However, due to time constraints this was not implemented in this study. Lastly, Table G.2 shows that for most cases, the scaling law give a lower error.

Table G.2: CH_4 critical temperature predictions in absolute error percentage.

Factor	Error [%]		
	3H	4H	SLRD
0.6	0.55	0.51	0.88
0.7	0.64	5.54	0.29
0.8	0.80	2.84	0.01
0.9	0.79	1.06	0.40
1.0	1.85	0.03	0.04

Table G.3 shows that for carbon dioxide the 4H network generally gives higher accuracy (lower error). In addition, here it can be seen that for almost all cases, more complete data sets lead to more accurate neural networks. Again, the scaling law score on average better than both networks.

Since the 3H network generally overpredicts the critical point for carbon dioxide, the fact that the 4H network somewhat flatten out there where no data is present in this case helps to increase its accuracy. This is shown in Figures G.4 and G.5.

Table G.3: CO_2 critical temperature predictions in absolute error percentage.

Factor	Error [%]		
	3H	4H	SLRD
0.6	1.80	2.02	0.12
0.7	1.50	1.06	0.31
0.8	1.22	0.23	0.30
0.9	0.90	0.14	0.23
1.0	0.53	0.36	0.01

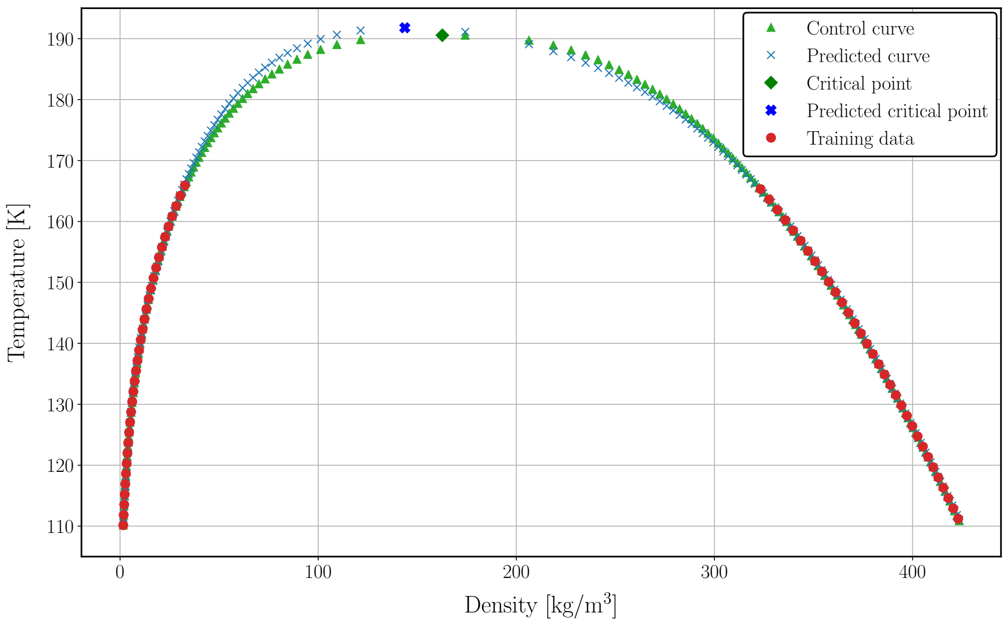


Figure G.2: Temperature-density coexistence curve for pure methane, comparison between values generated with REFPROP (green) and predicted using a neural network with **three** hidden neurons (blue). Red points show (a selection of) the data points on which the neural network was trained (factor = 0.7).

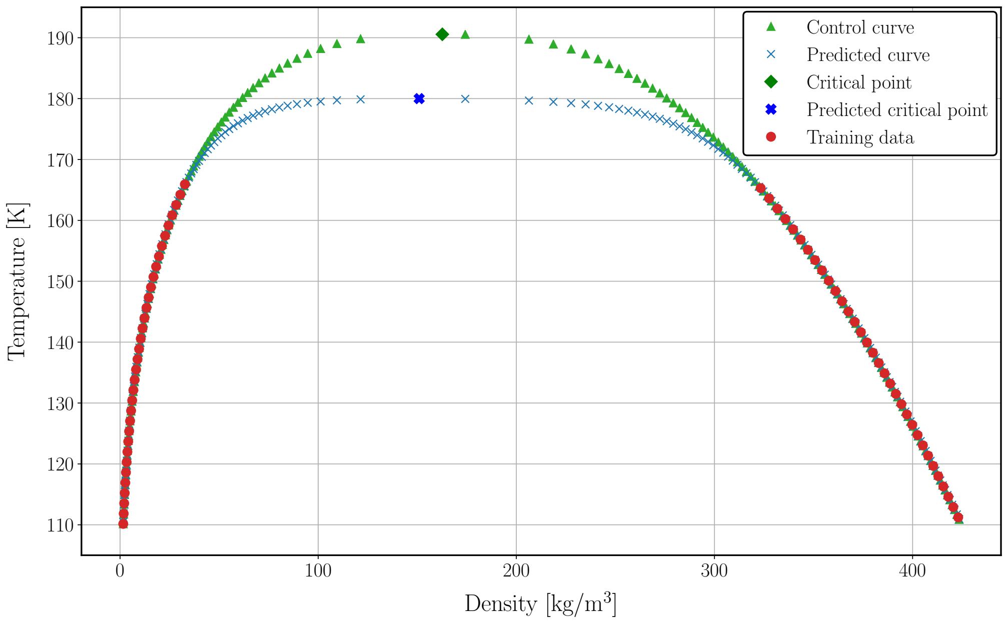


Figure G.3: Temperature-density coexistence curve for pure methane, comparison between values generated with REFPROP (green) and predicted using a neural network with **four** hidden neurons (blue). Red points show (a selection of) the data points on which the neural network was trained (factor = 0.7).

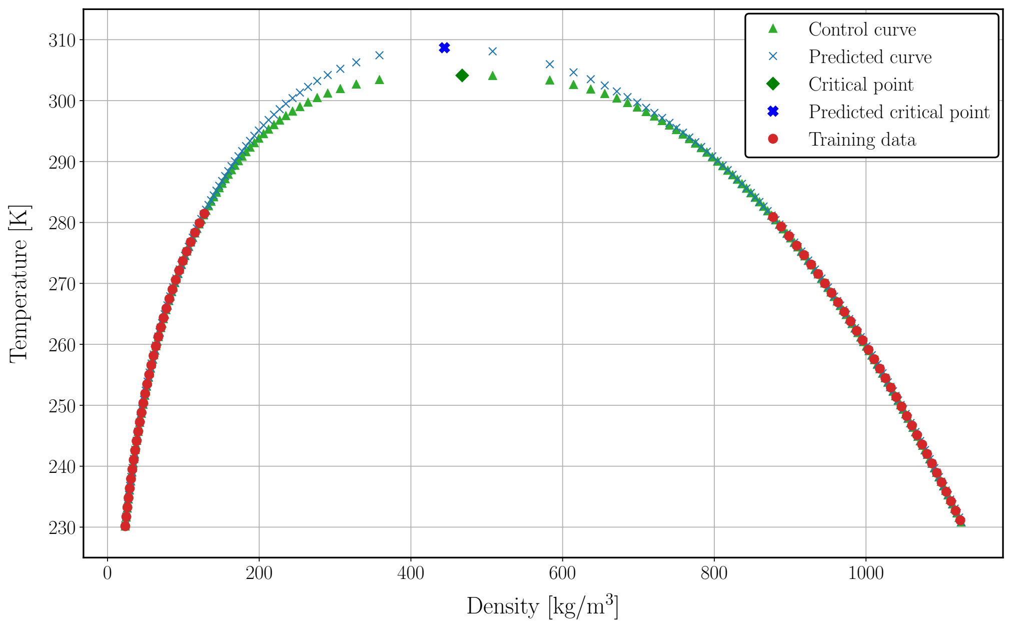


Figure G.4: Temperature-density coexistence curve for pure carbon dioxide, comparison between values generated with REFPROP (green) and predicted using a neural network with **three** hidden neurons (blue). Red points show (a selection of) the data points on which the neural network was trained (factor = 0.7).

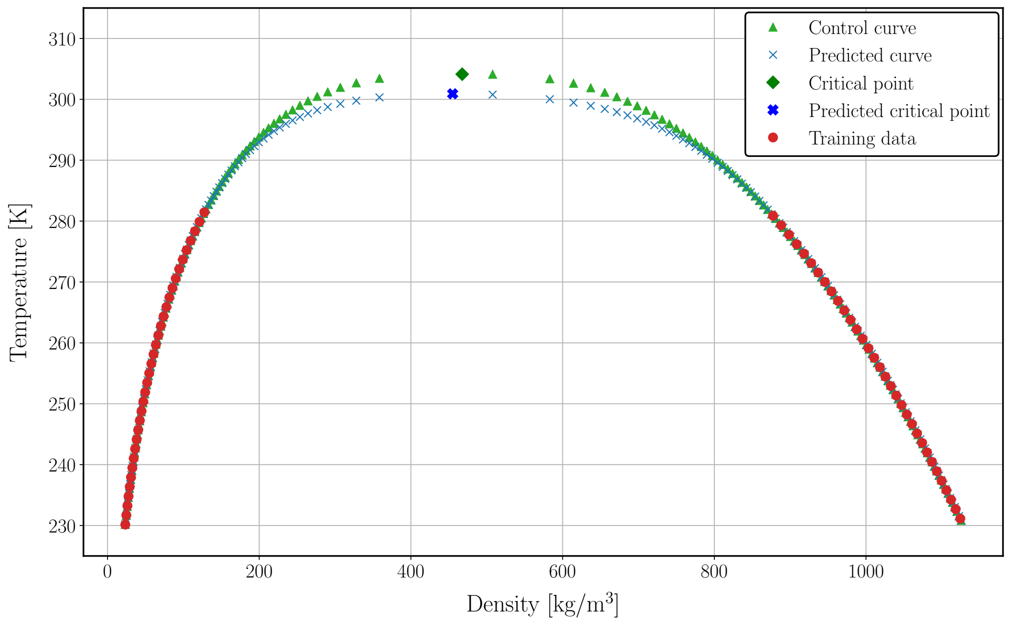


Figure G.5: Temperature-density coexistence curve for pure carbon dioxide, comparison between values generated with REFPROP (green) and predicted using a neural network with **four** hidden neurons (blue). Red points show (a selection of) the data points on which the neural network was trained (factor = 0.7).

G.2.2 Critical pressure

Table G.4 and G.5 show the accuracy of the critical pressure predictions for methane and carbon dioxide respectively. The tables compare results obtained using the neural network with three hidden neurons (3H), four hidden neurons (4H), and finally the scaling law of rectilinear diameter (SLDR).

From Table G.4 is can easily be seen that for methane, the 3H network again shows much higher accuracy scores than the 4H network. It can also be seen much more clearly how more complete data leads to much more accurate predictions. Again, the 3H network struggles to fit properly to the complete data set, most likely for the same reasons as previously given. In general, accuracy achieved using the scaling law is better or comparable (to within about a percentile point) to the 3H network, although it varies a lot (the reason for which is unknown). Figures G.6 and G.6 show the pressure-density coexistence curves and neural network prediction for the factor = 0.7 case.

Table G.4: CH_4 critical pressure predictions in absolute error percentage.

Factor	Error [%]		
	3H	4H	SLRD
0.6	9.92	48.79	0.29
0.7	5.18	37.05	3.73
0.8	2.01	24.72	3.10
0.9	1.08	12.68	1.68
1.0	20.21	2.39	1.37

Table G.5 shows that for the critical pressure prediction of carbon dioxide there is no clear network that performs better, as results vary quite a bit. Again the scaling law perform better overall. Figures G.8 and G.8 show the pressure-density coexistence curves and neural network prediction for the factor = 0.7 case.

Table G.5: CO_2 critical pressure predictions in absolute error percentage.

Factor	Error [%]		
	3H	4H	SLRD
0.6	9.42	17.16	2.00
0.7	10.29	10.48	2.95
0.8	2.55	4.63	3.08
0.9	6.17	6.12	2.71
1.0	6.07	2.76	0.66

It should also be noted that prediction for the critical pressure are in all cases much less accurate than predictions for the critical temperature, this goes for all networks for both compound and for the scaling law.

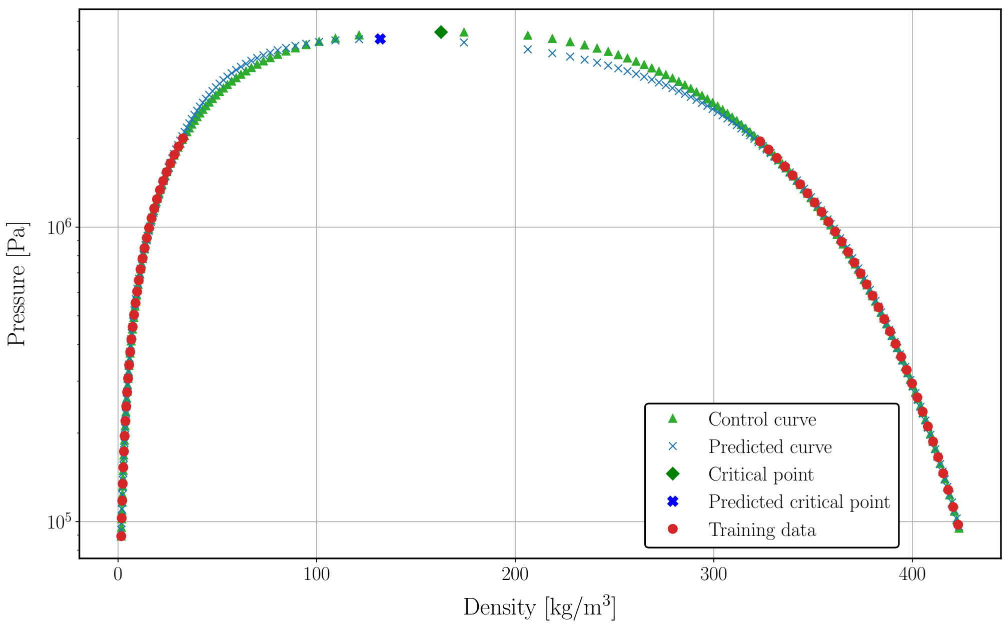


Figure G.6: Pressure-density coexistence curve for pure methane, comparison between values generated with REFPROP (green) and predicted using a neural network with **three** hidden neurons (blue). Red points show (a selection of) the data points on which the neural network was trained (factor = 0.7).

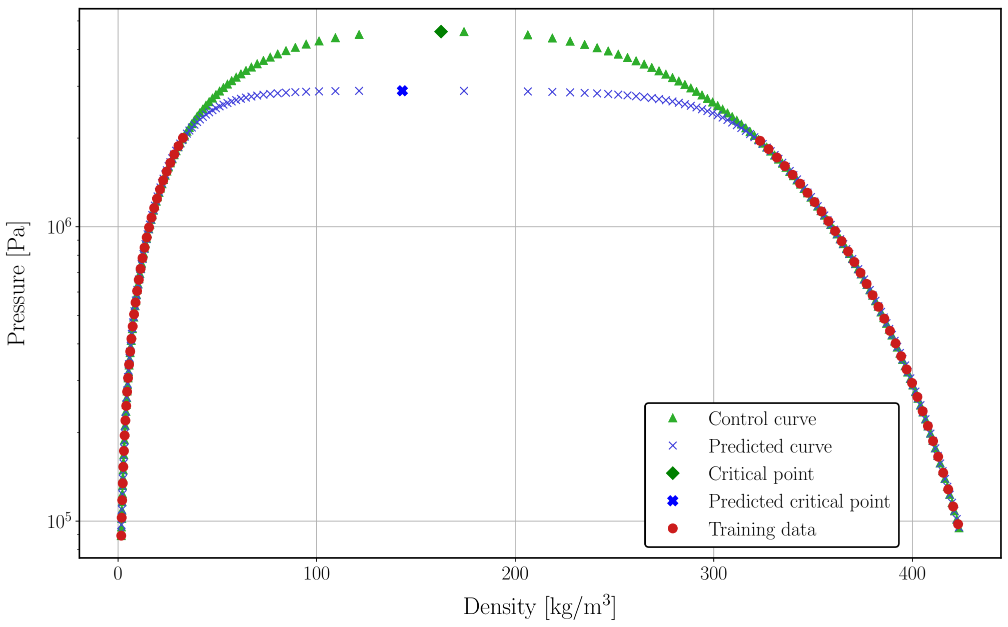


Figure G.7: Pressure-density coexistence curve for pure methane, comparison between values generated with REFPROP (green) and predicted using a neural network with **four** hidden neurons (blue). Red points show (a selection of) the data points on which the neural network was trained (factor = 0.7).

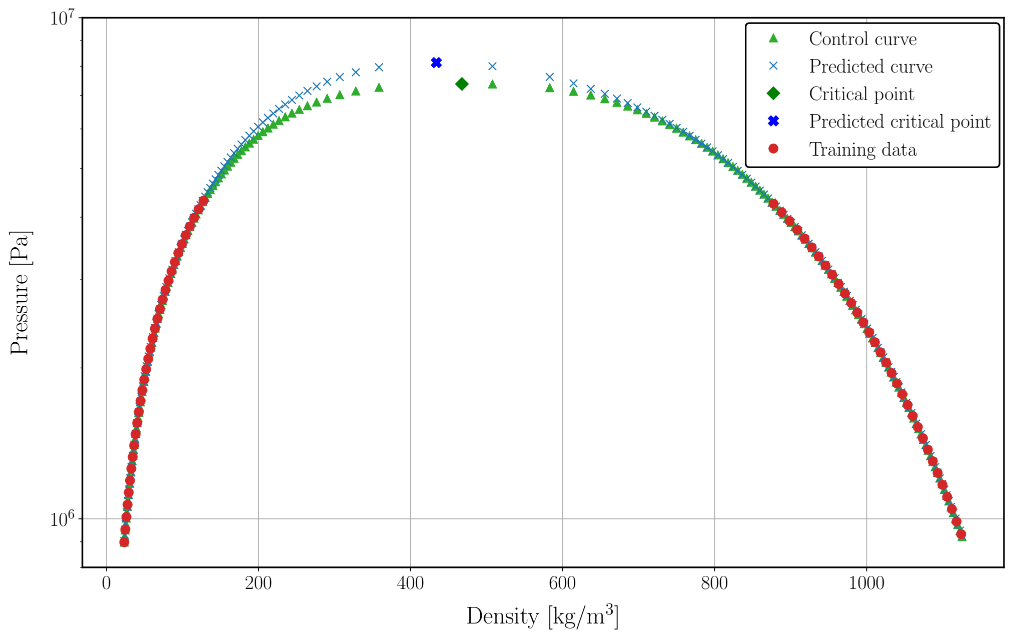


Figure G.8: Pressure-density coexistence curve for pure carbon dioxide, comparison between values generated with REFPROP (green) and predicted using a neural network with **three** hidden neurons (blue). Red points show (a selection of) the data points on which the neural network was trained (factor = 0.7).

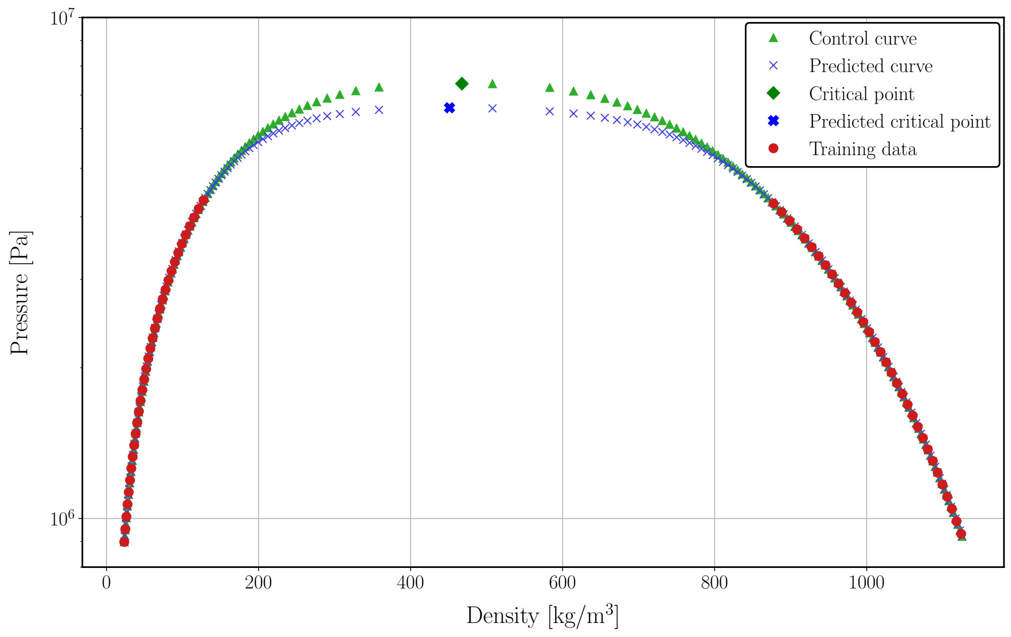


Figure G.9: Pressure-density coexistence curve for pure carbon dioxide, comparison between values generated with REFPROP (green) and predicted using a neural network with **three** hidden neurons (blue). Red points show (a selection of) the data points on which the neural network was trained (factor = 0.7).

G.2.3 Critical density

Table G.6 and G.7 show the accuracy of the critical density predictions for methane and carbon dioxide respectively. The tables compare results obtained using the neural network with three hidden neurons (3H), four hidden neurons (4H), and finally the scaling law of rectilinear diameter (SLDR).

As critical density was determined based on the density where either temperature or pressure took on its maximum value, for each network type there are two different critical density predictions, one corresponding to temperature and one to pressure, both are given in Tables G.6 and G.7.

Table G.6 shows that for methane the density corresponding to temperature predicted by the 4H network generally shows the highest accuracy of the neural networks, although again the scaling law are in most cases more accurate. Similar results are shown in Table G.7 for carbon dioxide, which also shows overall lower error scores than were seen for methane.

Table G.6: CH_4 critical density predictions in absolute error percentage.

Factor	Error [%]				
	Temperature		Pressure		
	3H	4H	3H	4H	SLRD
0.6	15.10	15.69	23.96	18.19	1.83
0.7	11.65	7.20	18.83	11.98	0.41
0.8	8.73	2.95	15.19	7.04	0.81
0.9	6.52	1.19	11.73	4.08	1.30
1.0	0.04	0.87	10.22	3.05	0.41

Table G.7: CO_2 critical density predictions in absolute error percentage.

Factor	Error [%]				
	Temperature		Pressure		
	3H	4H	3H	4H	SLRD
0.6	6.96	4.66	19.75	5.68	0.04
0.7	5.00	2.69	7.20	3.49	0.28
0.8	3.56	1.17	12.92	1.96	0.23
0.9	2.64	0.78	3.77	3.81	0.11
1.0	4.01	0.24	1.46	3.78	0.02

G.3 Final remarks

The aim of this small investigation was to look into whether neural networks could be used to make accurate predictions even when parts of the data to train on are missing. In order to make the problem more relatable to a real world problem (instead of just using meaningless dummy data points), pure compound coexistence curves were used as the data sets and made to not include the near-critical region in order to simulate gaps in data. Neural networks were trained on these partially complete data sets and made to predict the critical temperature, pressure, and density. For good measure, the results were compared to a conventional method of predicting the critical point based on incomplete data: the scaling law of rectilinear diameter.

Results from this investigation indicate that neural networks trained on incomplete data sets can still fairly accurately predict the missing data by “filling in” the gaps. The extend to which depends on the number of neurons in the hidden layer and the size of the data gap. While the scaling law in almost all cases gave a better prediction of the critical point, the power of the neural networks lies in the fact that they can fill in an entire gap, while the scaling law only give the value of a single point (the critical point). The current investigation only trained each neural network once, however, due to the stochastic nature of the training algorithm, each network should have been trained multiple times, after which the average results should have been reported. Due to time constraints this mistake could not be straightened up.

Appendix H

Accuracy scores excluding incorrect classifications

In this appendix the accuracy scores excluding incorrect classifications are given for all flashes and properties that were discussed in Section 5.2.3.

H.1 PT-flash

Table H.1: Pressure, temperature flash: **liquid** phase region accuracy scores for data sets excluding points of which the phase was incorrectly classified. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE_{sc}), and the coefficient of deviation (R^2). The mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are also shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE_{sc}	R^2
V [m^3/mol]	4.18e-05	2.24e-05	6.88e-07	3.08e+00	7.17e-03	9.923e-01
H [J/mol]	-2.79e+04	1.22e+04	1.94e+02	1.59e+00	3.28e-03	9.989e-01
S [J/mol/K]	-8.18e+01	2.69e+01	4.05e-01	1.51e+00	3.43e-03	9.992e-01
G [J/mol]	4.75e+03	7.95e+03	9.78e+01	1.23e+00	2.83e-03	9.997e-01
A [J/mol]	3.99e+03	7.58e+03	1.03e+02	1.36e+00	3.12e-03	9.996e-01
U [J/mol]	-2.87e+04	1.17e+04	1.26e+02	1.08e+00	2.29e-03	9.995e-01

Table H.2: Pressure, temperature flash: **vapor** phase region accuracy scores for data sets excluding points of which the phase was incorrectly classified. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE_{sc}), and the coefficient of deviation (R^2). The mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are also shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE_{sc}	R^2
V [m^3/mol]	2.51e-03	1.37e-02	2.07e-05	1.51e-01	3.91e-05	9.996
H [J/mol]	9.75e+03	4.31e+03	2.68e+02	6.22e+00	4.19e-03	9.887e-01
S [J/mol/K]	-4.05e+00	1.28e+01	4.11e-01	3.22e+00	2.30e-03	9.969e-01
G [J/mol]	1.23e+04	5.44e+03	1.50e+02	2.76e+00	3.17e-03	9.986e-01
A [J/mol]	8.28e+03	5.78e+03	1.12e+02	1.94e+00	2.26e-03	9.993e-01
U [J/mol]	5.69e+03	3.59e+03	2.42e+02	6.73e+00	4.03e-03	9.878e-01

Table H.3: Pressure, temperature flash: **two-phase** phase region accuracy scores for data sets excluding points of which the phase was incorrectly classified. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE_{sc}), and the coefficient of deviation (R^2). The mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are also shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE_{sc}	R^2
V [m^3/mol]	1.33e-03	5.63e-03	2.27e-04	4.03e+00	1.07e-03	9.662e-01
H [J/mol]	-9.37e+03	8.70e+03	1.11e+03	1.28e+01	2.36e-02	9.591e-01
S [J/mol/K]	-3.82e+01	1.80e+01	2.42e+00	1.35e+01	1.96e-02	9.598e-01
G [J/mol]	9.52e+03	4.24e+03	4.11e+01	9.69e-01	1.94e-03	9.998e-01
A [J/mol]	7.89e+03	4.34e+03	1.42e+02	3.26e+00	6.03e-03	9.976e-01
U [J/mol]	-1.10e+04	7.98e+03	8.57e+02	1.07e+01	1.98e-02	9.725e-01
β_{w} [mol/mol]	4.51e-01	2.75e-01	4.14e-02	1.51e+01	5.25e-02	9.470e-01
β_{m} [mol/mol]	5.97e-01	2.74e-01	5.83e-02	2.13e+01	6.68e-02	9.212e-01
x_{w} [mol/mol]	7.22e-01	1.99e-01	3.61e-02	1.81e+01	5.79e-02	7.949e-01
y_{w} [mol/mol]	5.71e-01	2.07e-01	4.46e-02	2.16e+01	5.95e-02	6.900e-01

H.2 PS-flash

Table H.4: Pressure, entropy flash: **liquid** phase region accuracy scores for data sets excluding points of which the phase was incorrectly classified. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE_{sc}), and the coefficient of deviation (R^2). The mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are also shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE_{sc}	R^2
T [K]	4.31e+02	9.77e+01	1.10e+00	1.13e+00	2.76e-03	9.996e-01
V [m^3/mol]	4.18e-05	2.23e-05	5.05e-07	2.27e+00	5.74e-03	9.940e-01
H [J/mol]	-2.80e+04	1.22e+04	1.25e+02	1.03e+00	2.10e-03	9.998e-01
G [J/mol]	4.74e+03	7.94e+03	1.26e+02	1.59e+00	3.67e-03	9.995e-01
A [J/mol]	3.98e+03	7.58e+03	1.52e+02	2.01e+00	4.60e-03	9.993e-01
U [J/mol]	-2.87e+04	1.17e+04	8.92e+01	7.62e-01	1.60e-03	9.999e-01

Table H.5: Pressure, entropy flash: **vapor** phase region accuracy scores for data sets excluding points of which the phase was incorrectly classified. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE_{sc}), and the coefficient of deviation (R^2). The mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are also shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE_{sc}	R^2
T [K]	6.19e+02	6.24e+01	2.93e+00	4.70e+00	7.13e-03	9.944e-01
V [m^3/mol]	2.51e-03	1.37e-02	4.39e-05	3.21e-01	8.26e-05	9.993e-01
H [J/mol]	9.75e+03	4.31e+03	1.30e+02	3.00e+00	1.98e-03	9.983e-01
G [J/mol]	1.23e+04	5.44e+03	1.85e+02	3.41e+00	3.92e-03	9.976e-01
A [J/mol]	8.28e+03	5.78e+03	1.65e+02	2.86e+00	3.32e-03	9.981e-01
U [J/mol]	5.69e+03	3.60e+03	2.25e+02	6.25e+00	3.69e-03	9.934e-01

Table H.6: Pressure, entropy flash: **two-phase** phase region accuracy scores for data sets excluding points of which the phase was incorrectly classified. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). The mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are also shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
T [K]	5.00e+02	5.87e+01	7.31e-01	1.25e+00	3.10e-03	9.997e-01
V [m^3/mol]	1.28e-03	5.51e-03	2.55e-05	4.63e-01	1.30e-04	9.995e-01
H [J/mol]	-9.34e+03	8.65e+03	1.11e+02	1.28e+00	2.19e-03	9.996e-01
G [J/mol]	9.71e+03	4.32e+03	6.81e+01	1.58e+00	3.07e-03	9.995e-01
A [J/mol]	8.09e+03	4.39e+03	9.61e+01	2.19e+00	3.90e-03	9.990e-01
U [J/mol]	-1.10e+04	7.95e+03	8.13e+01	1.02e+00	1.77e-03	9.997e-01
β_{w} [mol/mol]	4.53e-01	2.71e-01	8.23e-03	3.04e+00	1.01e-02	9.960e-01
β_{m} [mol/mol]	5.97e-01	2.69e-01	1.38e-02	5.13e+00	1.54e-02	9.913e-01
x_{w} [mol/mol]	7.18e-01	2.07e-01	9.80e-03	4.74e+00	2.67e-02	9.725e-01
y_{w} [mol/mol]	5.72e-01	2.14e-01	1.64e-02	7.67e+00	3.10e-02	9.032e-01

H.3 SV-flash

Table H.7: Entropy, volume flash: **liquid** phase region accuracy scores for data sets excluding points of which the phase was incorrectly classified. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). The mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are also shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
P [Pa]	1.70e+07	8.28e+06	1.58e+07	1.90e+02	5.25e-01	-6.170e+00
T [K]	4.31e+02	9.81e+01	2.00e+00	2.04e+00	4.92e-03	9.979e-01
H [J/mol]	-2.79e+04	1.22e+04	3.61e+02	2.95e+00	6.02e-03	9.986e-01
G [J/mol]	4.76e+03	7.95e+03	2.78e+02	3.49e+00	8.04e-03	9.975e-01
A [J/mol]	4.00e+03	7.59e+03	1.45e+02	1.91e+00	4.34e-03	9.993e-01
U [J/mol]	-2.87e+04	1.18e+04	8.13e+01	6.91e-01	1.45e-03	9.999e-01

Table H.8: Entropy, volume flash: **vapor** phase region accuracy scores for data sets excluding points of which the phase was incorrectly classified. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). The mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are also shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
P [Pa]	1.05e+07	7.78e+06	5.96e+05	7.66e+00	1.99e-02	9.785e-01
T [K]	6.19e+02	6.23e+01	8.77e-01	1.41e+00	2.12e-03	9.996e-01
H [J/mol]	9.72e+03	4.33e+03	1.96e+02	4.53e+00	2.97e-03	9.961e-01
G [J/mol]	1.23e+04	5.44e+03	2.56e+02	4.71e+00	5.39e-03	9.955e-01
A [J/mol]	8.29e+03	5.78e+03	1.63e+02	2.83e+00	3.24e-03	9.984e-01
U [J/mol]	5.67e+03	3.61e+03	1.12e+02	3.11e+00	1.83e-03	9.981e-01

Table H.9: Entropy, volume flash: **two-phase** phase region accuracy scores for data sets excluding points of which the phase was incorrectly classified. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). The mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are also shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
P [Pa]	5.29e+06	3.95e+06	6.81e+04	1.72e+00	6.03e-02	9.993e-01
T [K]	4.99e+02	5.90e+01	6.30e-01	1.07e+00	6.91e-03	9.997e-01
H [J/mol]	-9.10e+03	8.76e+03	7.24e+01	8.26e-01	2.55e-03	9.998e-01
G [J/mol]	9.61e+03	4.34e+03	6.59e+01	1.52e+00	9.62e-03	9.995e-01
A [J/mol]	7.95e+03	4.39e+03	7.51e+01	1.71e+00	6.30e-03	9.994e-01
U [J/mol]	-1.08e+04	8.06e+03	1.14e+02	1.42e+00	2.38e-03	9.996e-01
β_{w} [mol/mol]	4.68e-01	2.69e-01	1.15e-02	4.28e+00	1.58e-02	9.951e-01
β_{m} [mol/mol]	6.15e-01	2.60e-01	1.25e-02	4.82e+00	1.78e-02	9.926e-01
x_{w} [mol/mol]	7.24e-01	2.04e-01	1.13e-02	5.55e+00	2.55e-02	9.523e-01
y_{w} [mol/mol]	5.76e-01	2.13e-01	8.26e-03	3.88e+00	3.74e-02	9.837e-01

H.4 HV-flash

Table H.10: Enthalpy, volume flash: **liquid** phase region accuracy scores for data sets excluding points of which the phase was incorrectly classified. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). The mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are also shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
P [Pa]	1.70e+07	8.28e+06	2.01e+07	2.43e+02	6.72e-01	-6.403e+01
T [K]	4.31e+02	9.82e+01	2.03e+00	2.06e+00	4.89e-03	9.985e-01
S [J/mol/K]	-8.17e+01	2.69e+01	8.85e-01	3.29e+00	7.40e-03	9.982e-01
G [J/mol]	4.77e+03	7.96e+03	3.10e+02	3.89e+00	8.91e-03	9.970e-01
A [J/mol]	4.01e+03	7.59e+03	1.51e+02	1.99e+00	4.50e-03	9.993e-01
U [J/mol]	-2.87e+04	1.18e+04	3.13e+02	2.66e+00	5.55e-03	9.989e-01

Table H.11: Enthalpy, volume flash: **vapor** phase region accuracy scores for data sets excluding points of which the phase was incorrectly classified. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\text{sc}}$), and the coefficient of deviation (R^2). The mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are also shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\text{sc}}$	R^2
P [Pa]	1.05e+07	7.78e+06	5.12e+05	6.59e+00	1.71e-02	9.826e-01
T [K]	6.19e+02	6.23e+01	1.25e+00	2.01e+00	3.03e-03	9.991e-01
S [J/mol/K]	-4.11e+00	1.28e+01	4.20e-01	3.28e+00	2.27e-03	9.975e-01
G [J/mol]	1.23e+04	5.44e+03	8.58e+01	1.58e+00	1.81e-03	9.996e-01
A [J/mol]	8.29e+03	5.78e+03	1.82e+02	3.16e+00	3.62e-03	9.980e-01
U [J/mol]	5.67e+03	3.61e+03	1.15e+02	3.20e+00	1.88e-03	9.980e-01

Table H.12: Enthalpy, volume flash: **two-phase** phase region accuracy scores for data sets excluding points of which the phase was incorrectly classified. Accuracy scores shown are: the mean absolute error (MAE), the MAE as a percentage of the standard deviation (MAE%), the MAE score for the same data set normalized to the interval [0, 1] (MAE$_{\mathrm{sc}}$), and the coefficient of deviation (R^2). The mean value and standard deviation of the test data sets (μ_{test} and σ_{test}) are also shown.

Property	μ_{test}	σ_{test}	MAE	MAE%	MAE$_{\mathrm{sc}}$	R^2
P [Pa]	5.25e+06	3.94e+06	5.43e+04	1.38e+00	1.71e-02	9.996e-01
T [K]	4.99e+02	5.90e+01	1.57e+00	2.66e+00	7.78e-03	9.988e-01
S [J/mol/K]	-3.82e+01	1.79e+01	4.72e-01	2.64e+00	3.96e-03	9.989e-01
G [J/mol]	9.61e+03	4.35e+03	2.73e+02	6.29e+00	1.14e-02	9.922e-01
A [J/mol]	7.96e+03	4.40e+03	1.59e+02	3.61e+00	7.30e-03	9.978e-01
U [J/mol]	-1.09e+04	8.04e+03	2.30e+02	2.86e+00	5.20e-03	9.988e-01
β_{w} [mol/mol]	4.60e-01	2.68e-01	6.64e-03	2.48e+00	1.34e-02	9.977e-01
β_{m} [mol/mol]	6.09e-01	2.61e-01	1.13e-02	4.33e+00	1.81e-02	9.952e-01
x_{w} [mol/mol]	7.27e-01	2.03e-01	7.87e-03	3.87e+00	2.88e-02	9.482e-01
y_{w} [mol/mol]	5.78e-01	2.13e-01	8.87e-03	4.16e+00	3.56e-02	9.534e-01

Appendix I

Selected correlation plots

I.1 PT-flash regression network correlation plots

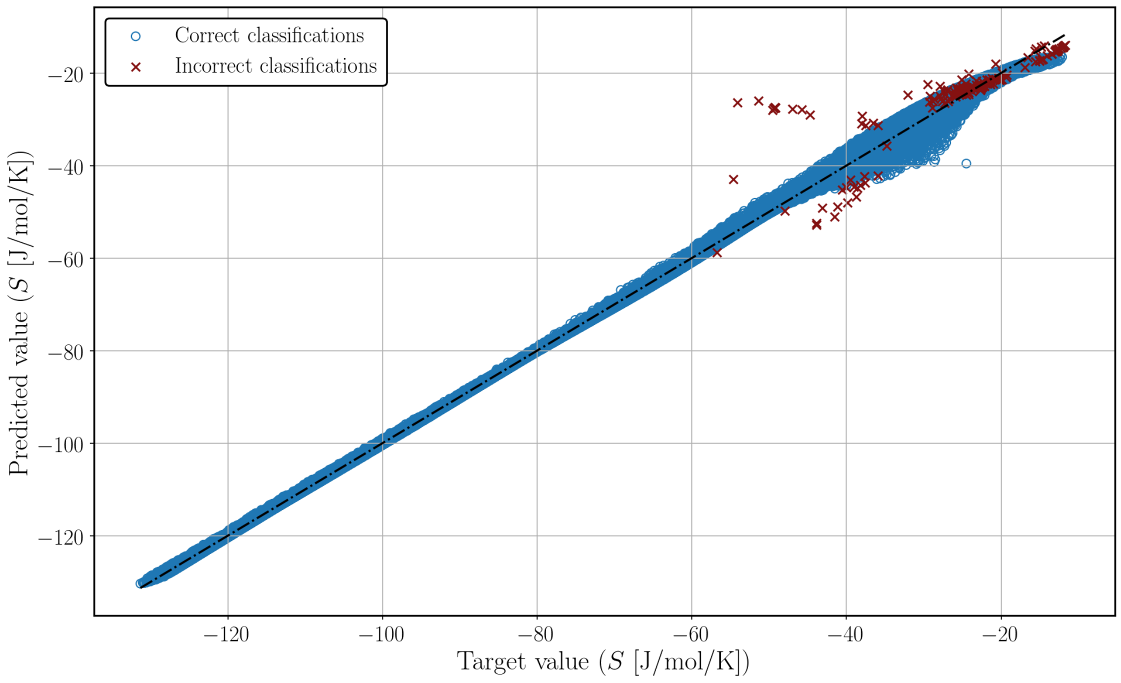


Figure I.1: Pressure-temperature flash: liquid entropy prediction correlation plot.

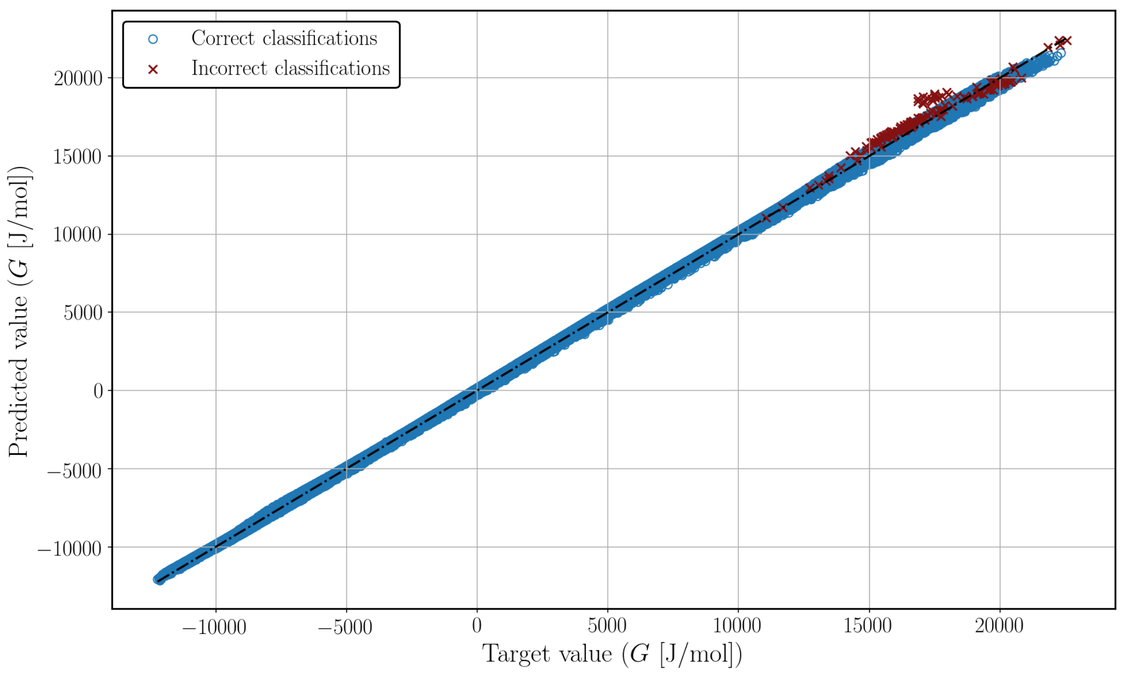


Figure I.2: Pressure-temperature flash: liquid Gibbs free energy prediction correlation plot.

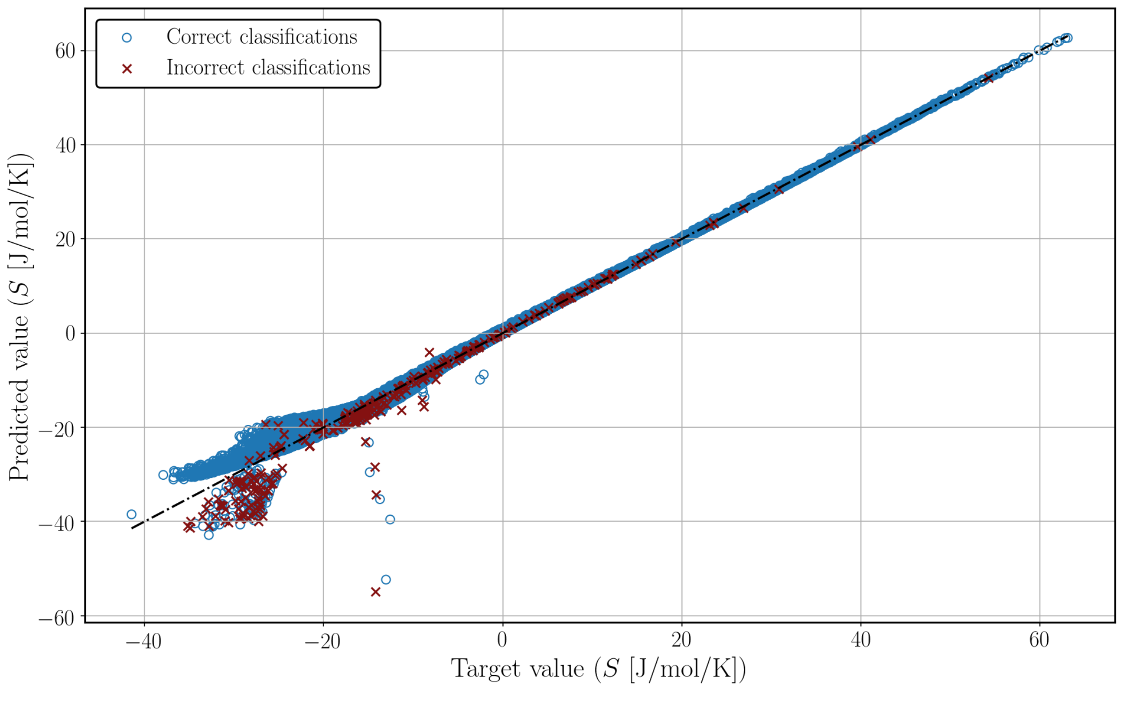


Figure I.3: Pressure-temperature flash: vapor entropy prediction correlation plot.

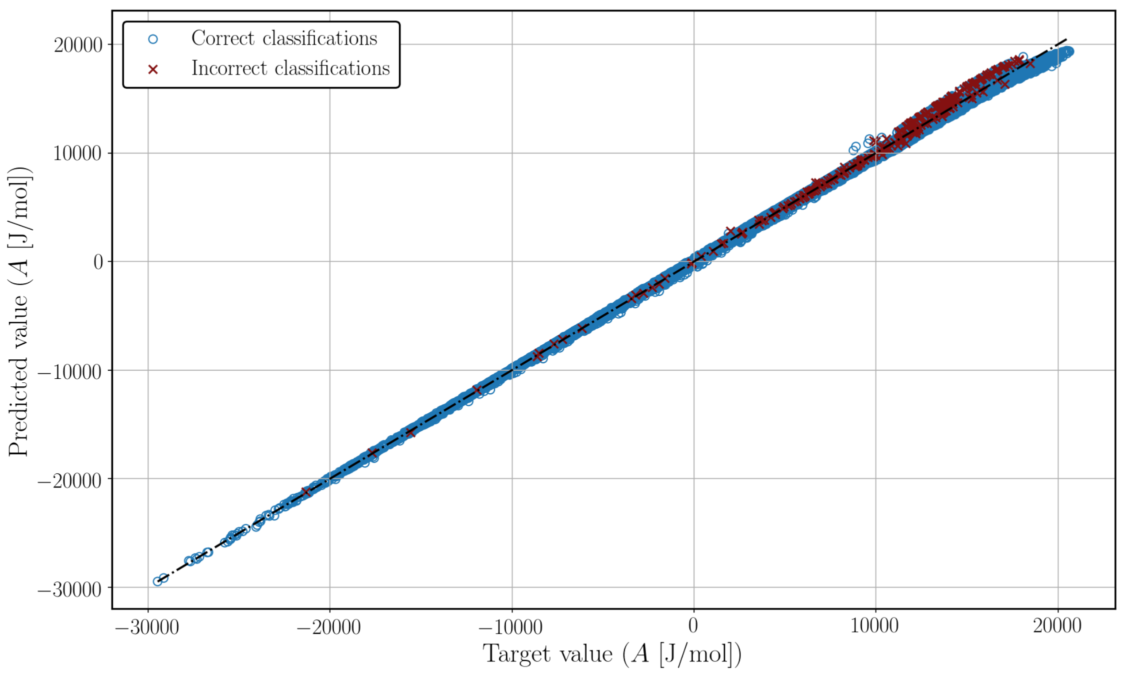


Figure I.4: Pressure-temperature flash: vapor Helmholtz free energy prediction correlation plot.

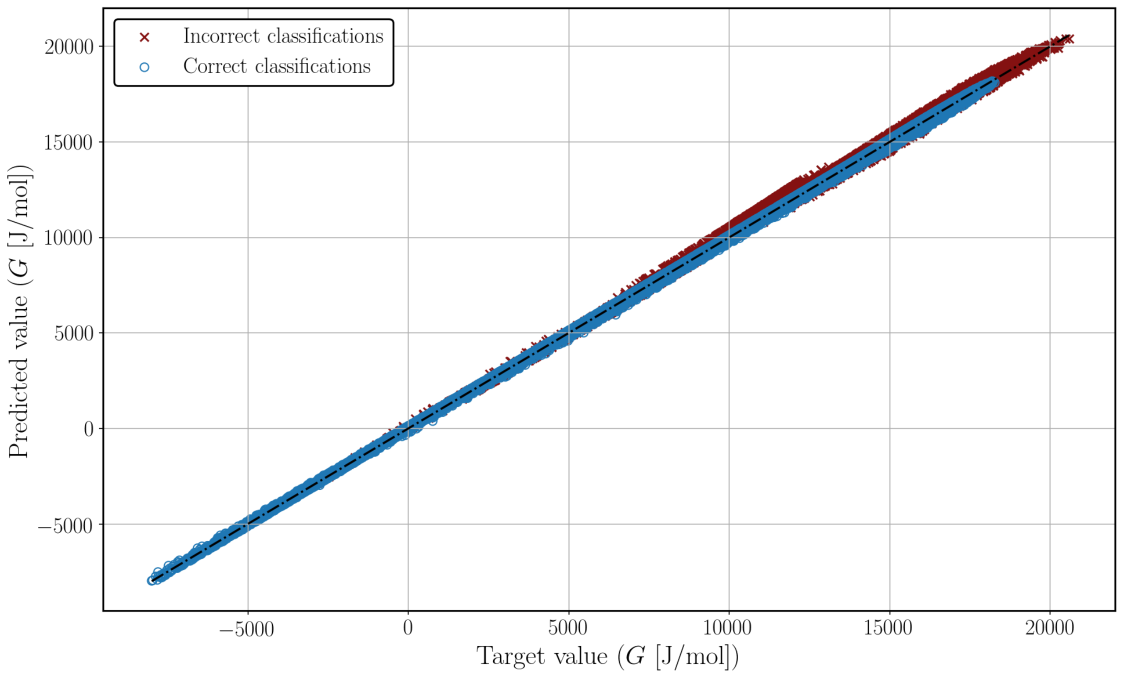


Figure I.5: Pressure-temperature flash: two-phase Gibbs free energy prediction correlation plot.

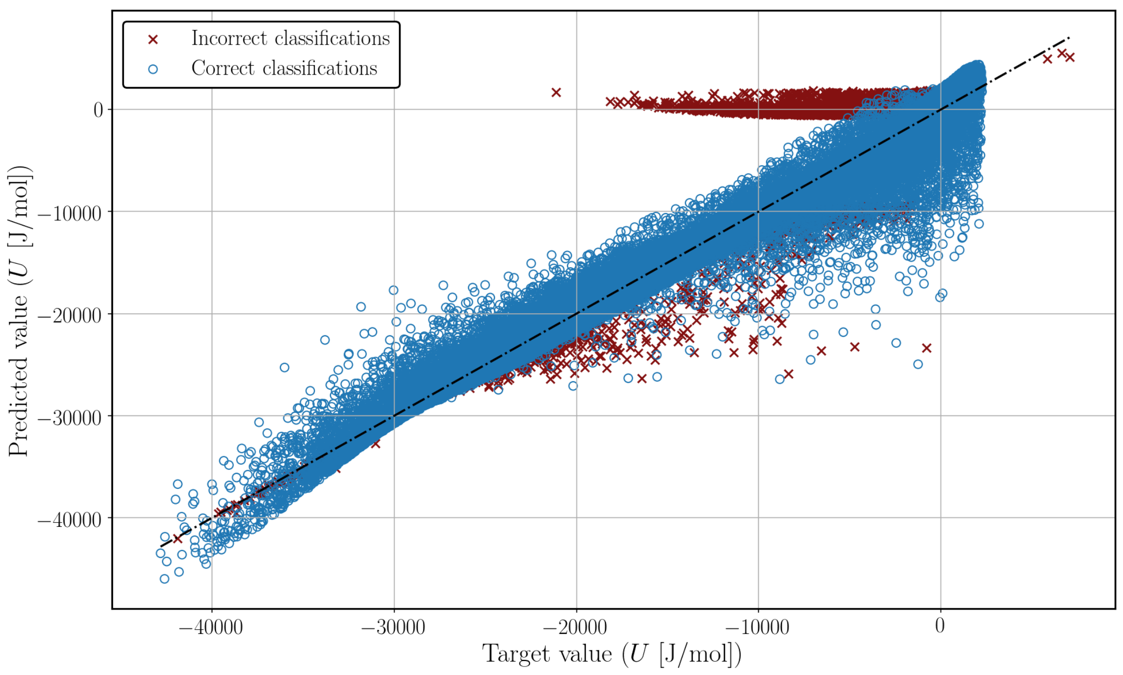


Figure I.6: Pressure-temperature flash: two-phase internal energy prediction correlation plot.

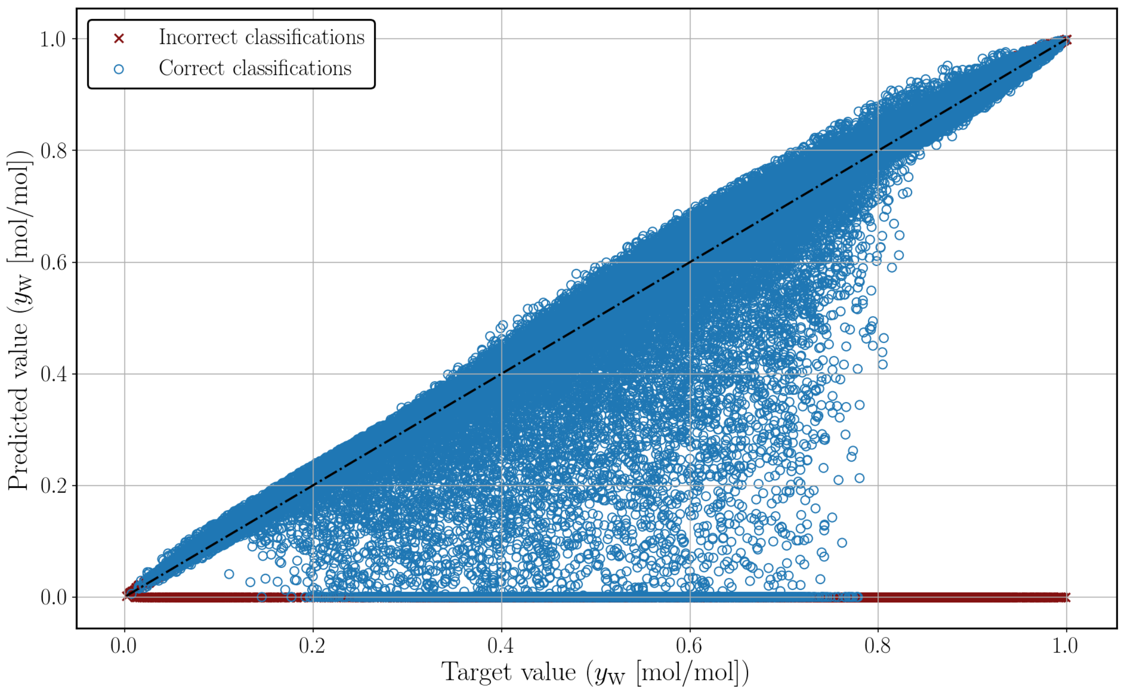


Figure I.7: Pressure-temperature flash: two-phase vapor phase mole fraction (water) prediction correlation plot.

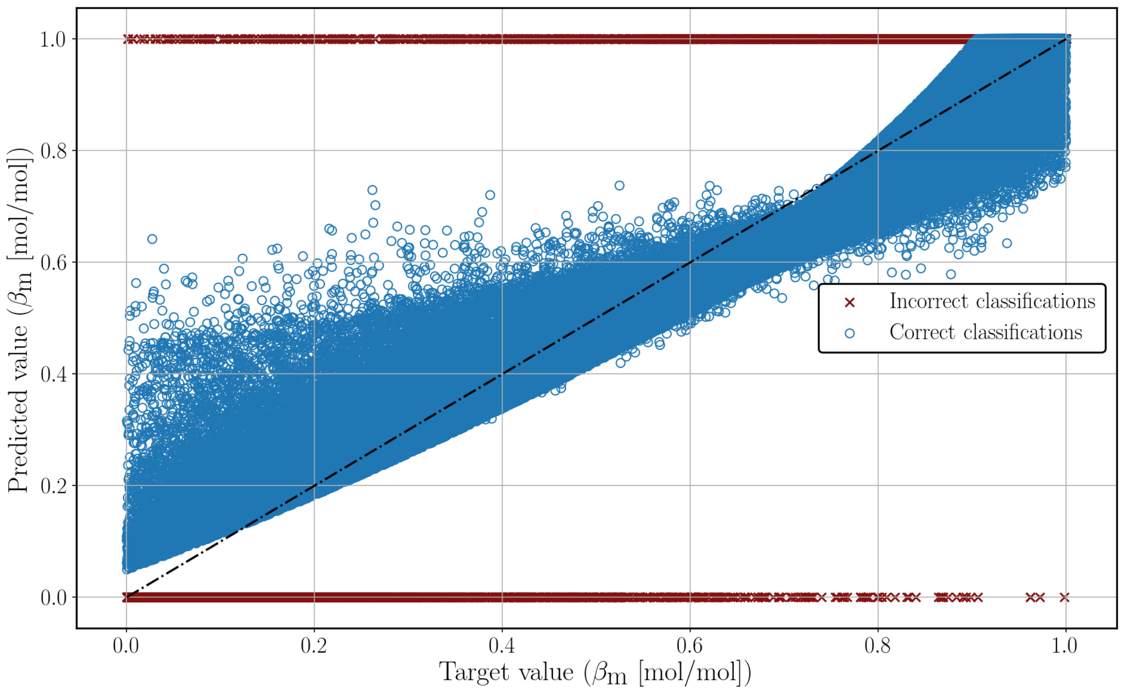


Figure I.8: Pressure-temperature flash: two-phase methanol vapor fraction prediction correlation plot.

I.2 PS-flash regression network correlation plots

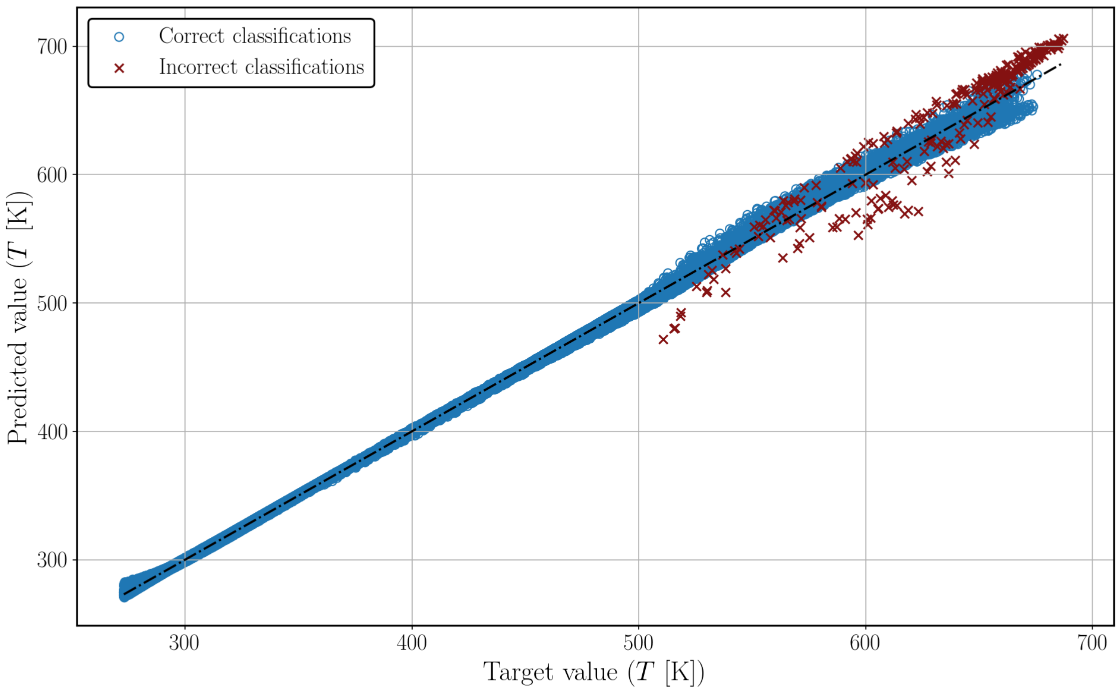


Figure I.9: Pressure-entropy flash: liquid temperature prediction correlation plot.

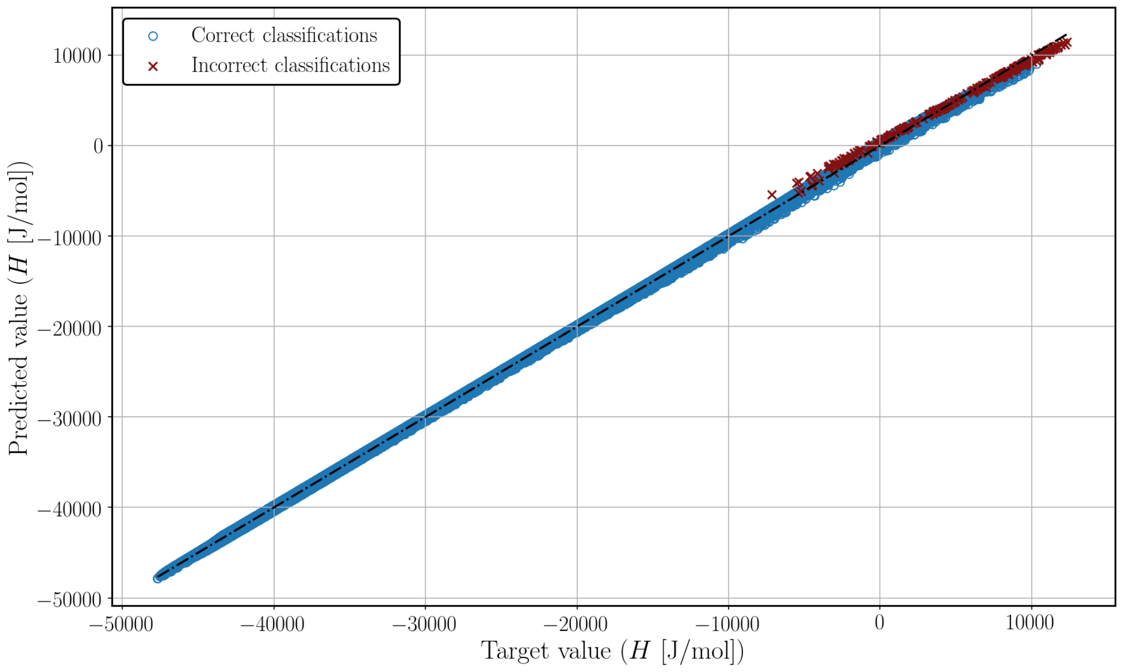


Figure I.10: Pressure-entropy flash: liquid enthalpy prediction correlation plot.

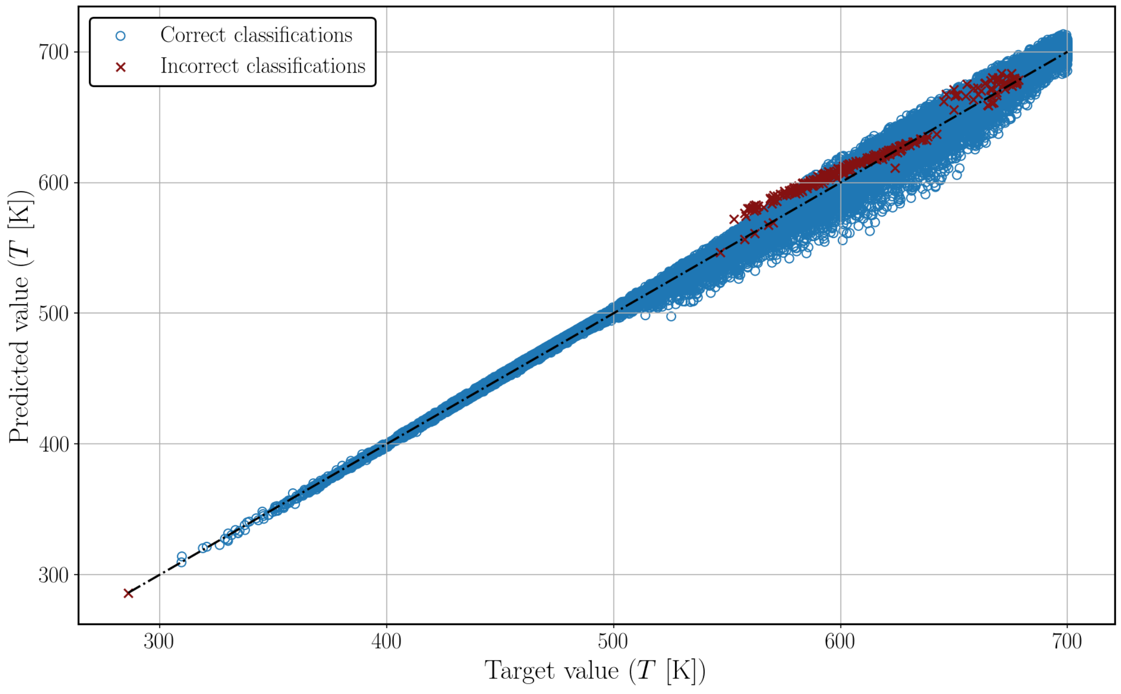


Figure I.11: Pressure-entropy flash: vapor temperature prediction correlation plot.

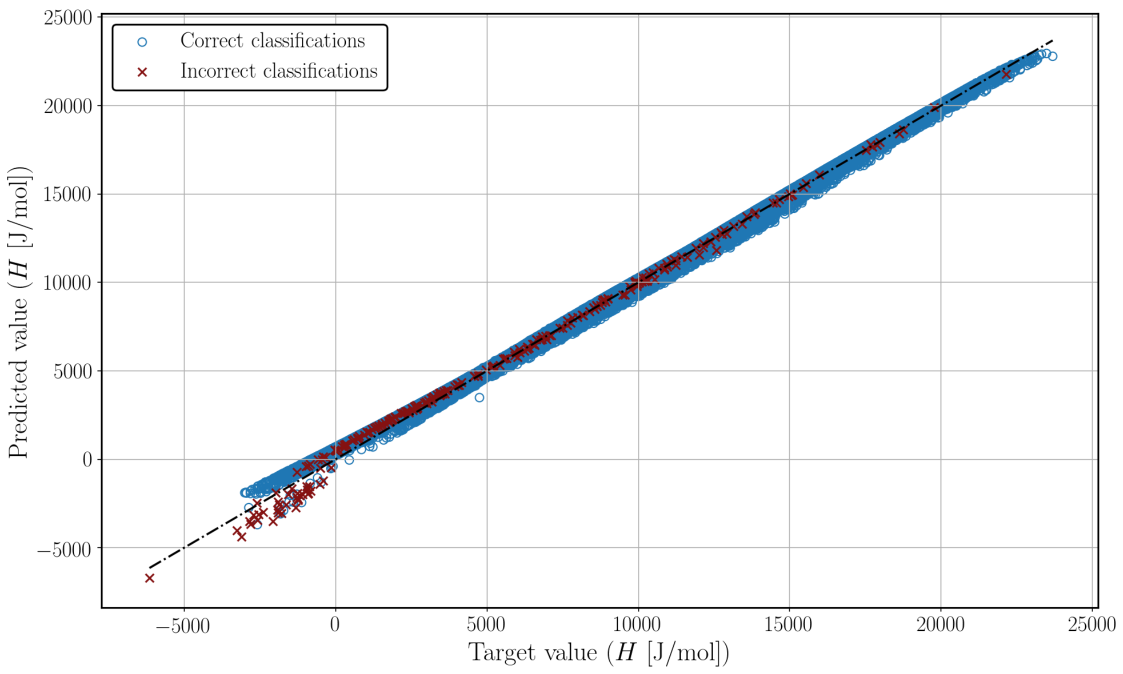


Figure I.12: Pressure-entropy flash: vapor enthalpy prediction correlation plot.

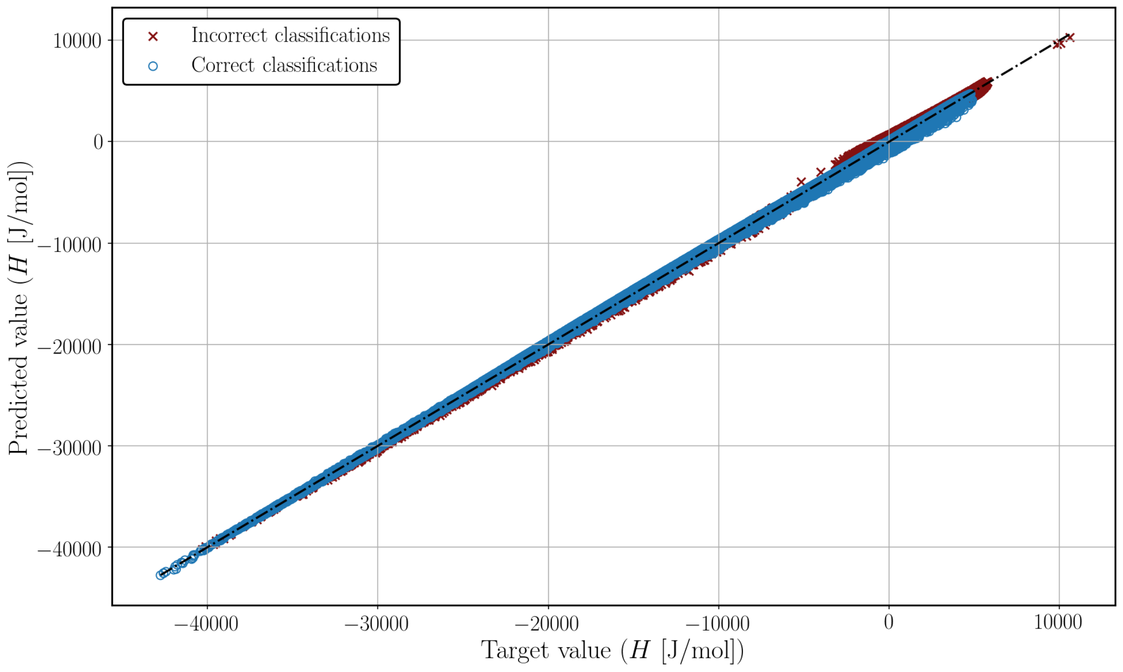


Figure I.13: Pressure-entropy flash: two-phase enthalpy prediction correlation plot.

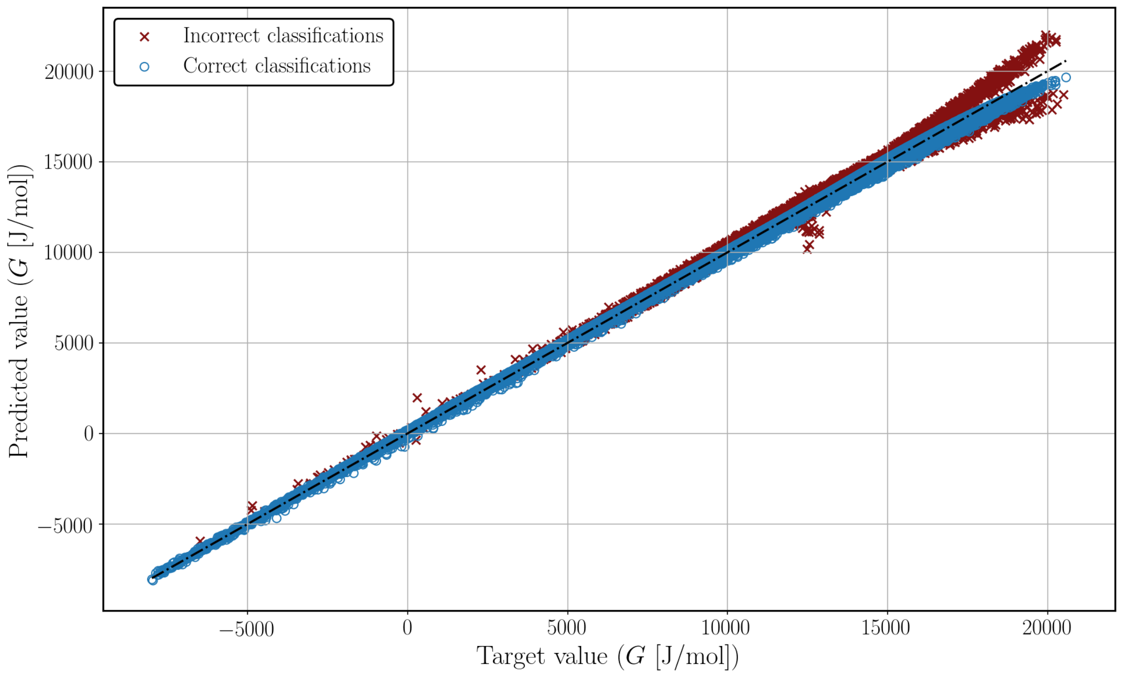


Figure I.14: Pressure-entropy flash: two-phase Gibbs free energy prediction correlation plot.

I.3 SV-flash regression network correlation plots

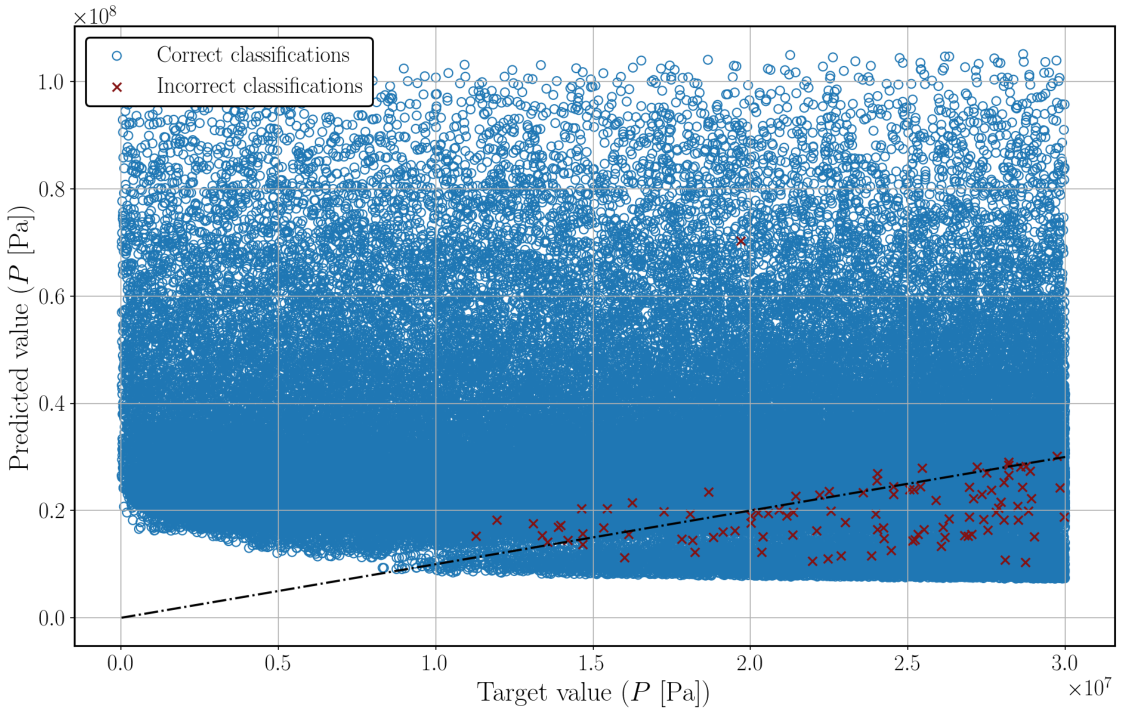


Figure I.15: Entropy-volume flash: liquid pressure prediction correlation plot.

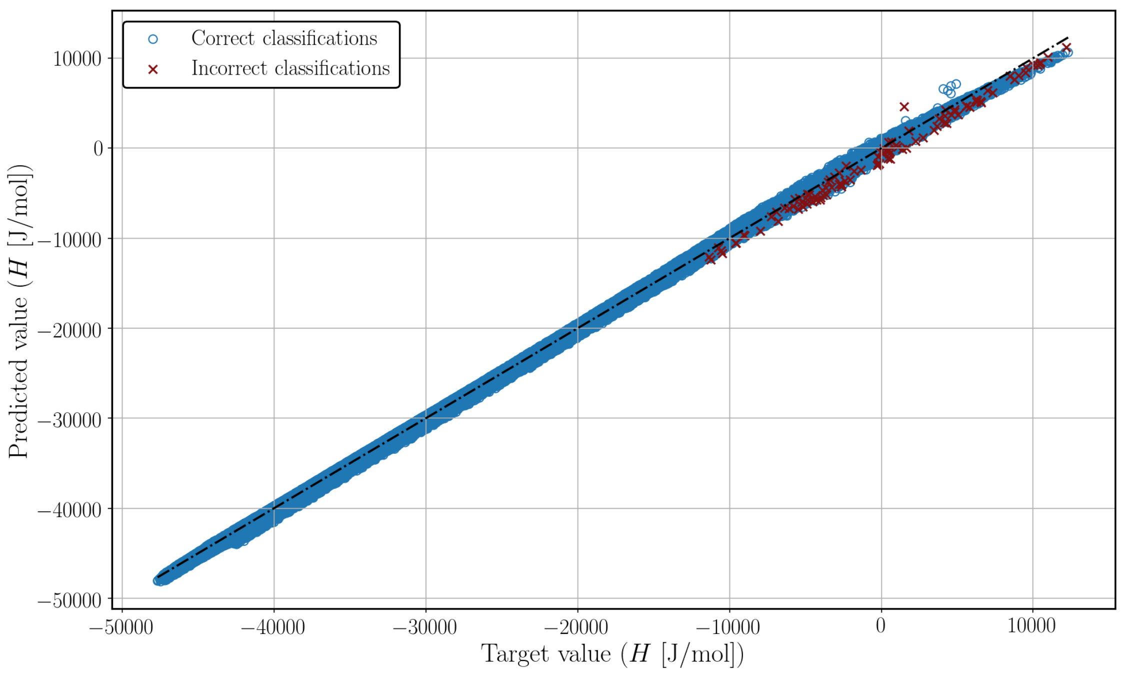


Figure I.16: Entropy-volume flash: liquid enthalpy prediction correlation plot.

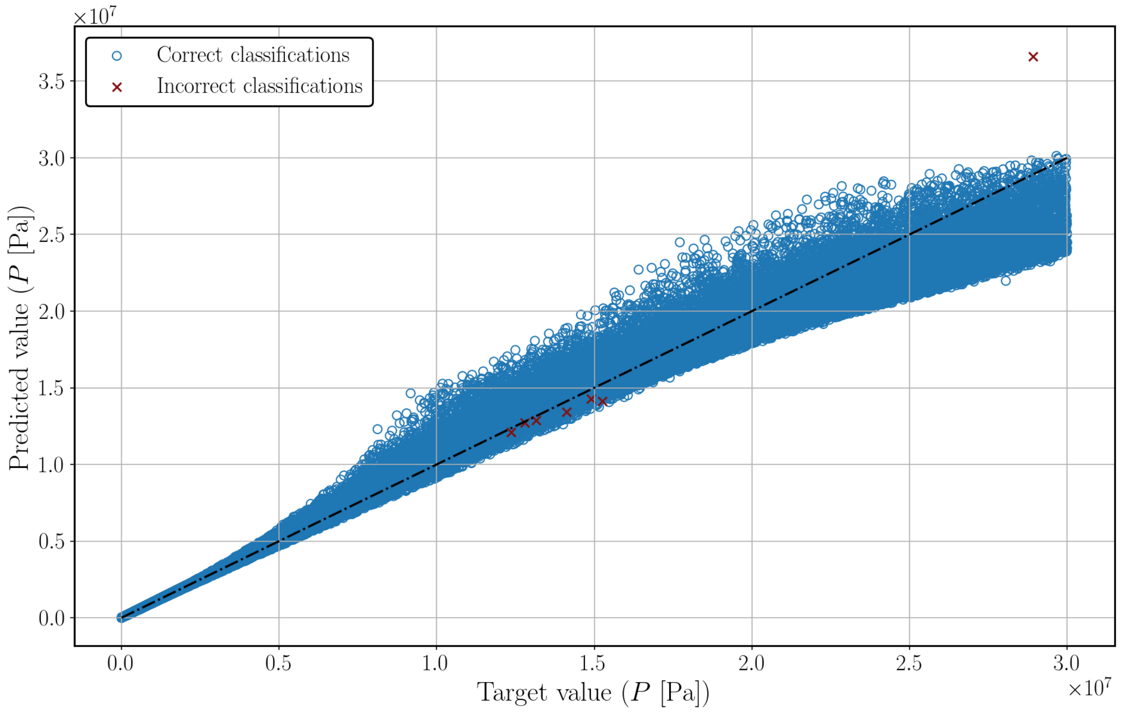


Figure I.17: Entropy-volume flash: vapor pressure prediction correlation plot.

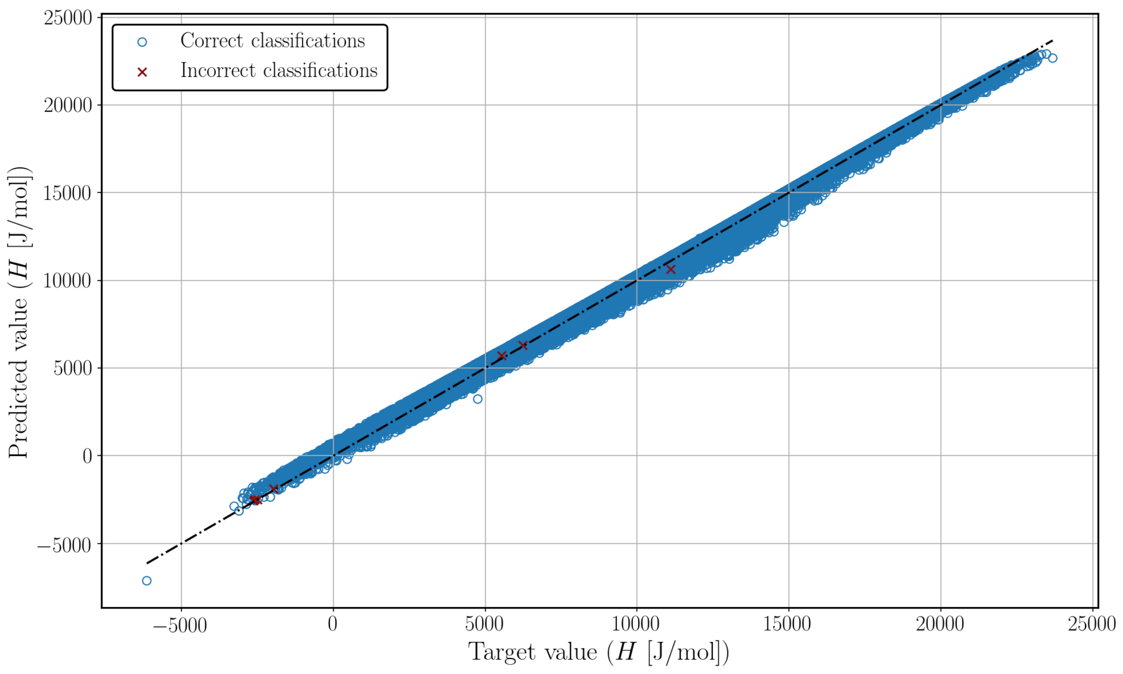


Figure I.18: Entropy-volume flash: vapor enthalpy prediction correlation plot.

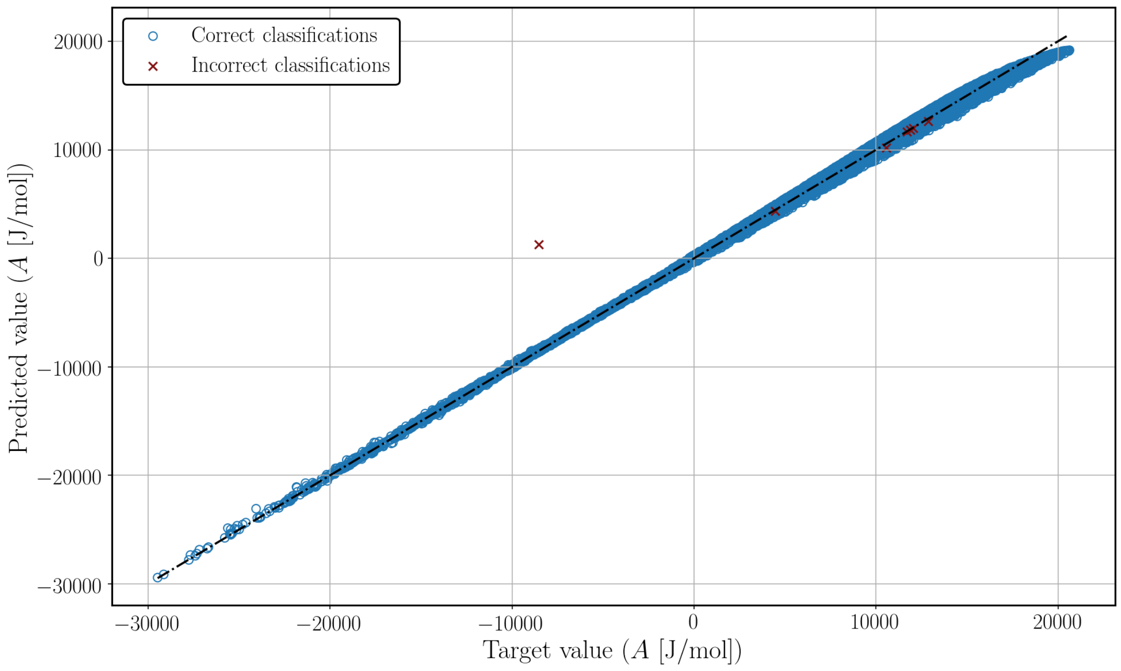


Figure I.19: Entropy-volume flash: vapor Helmholtz free energy prediction correlation plot.

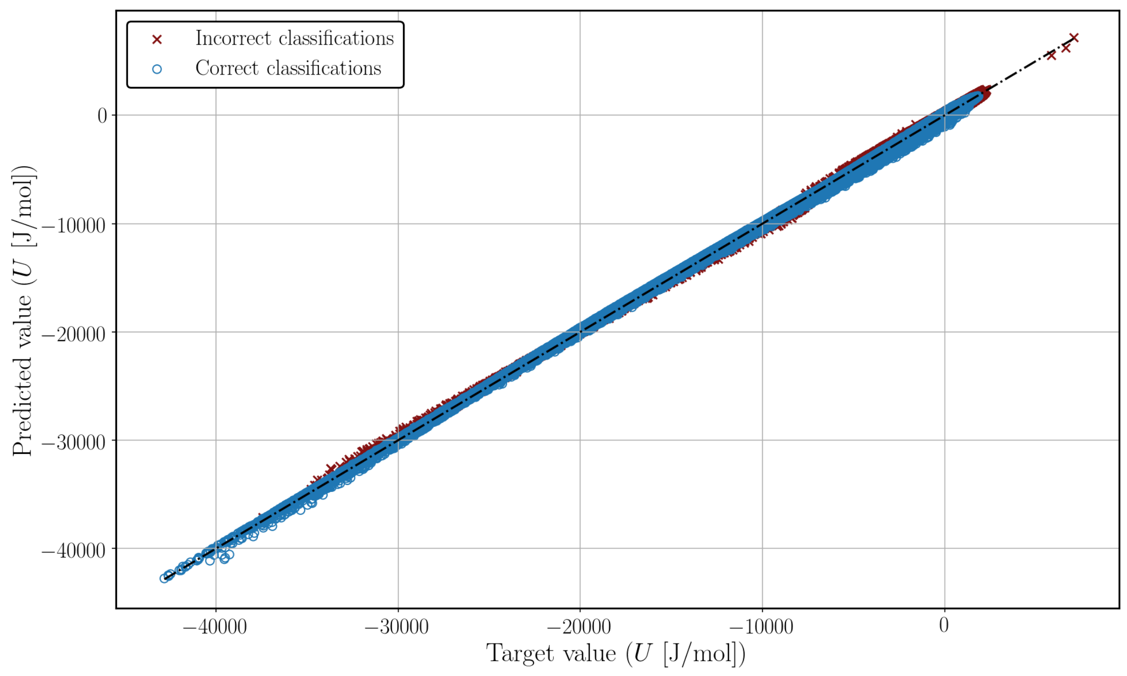


Figure I.20: Entropy-volume flash: two-phase internal energy prediction correlation plot.

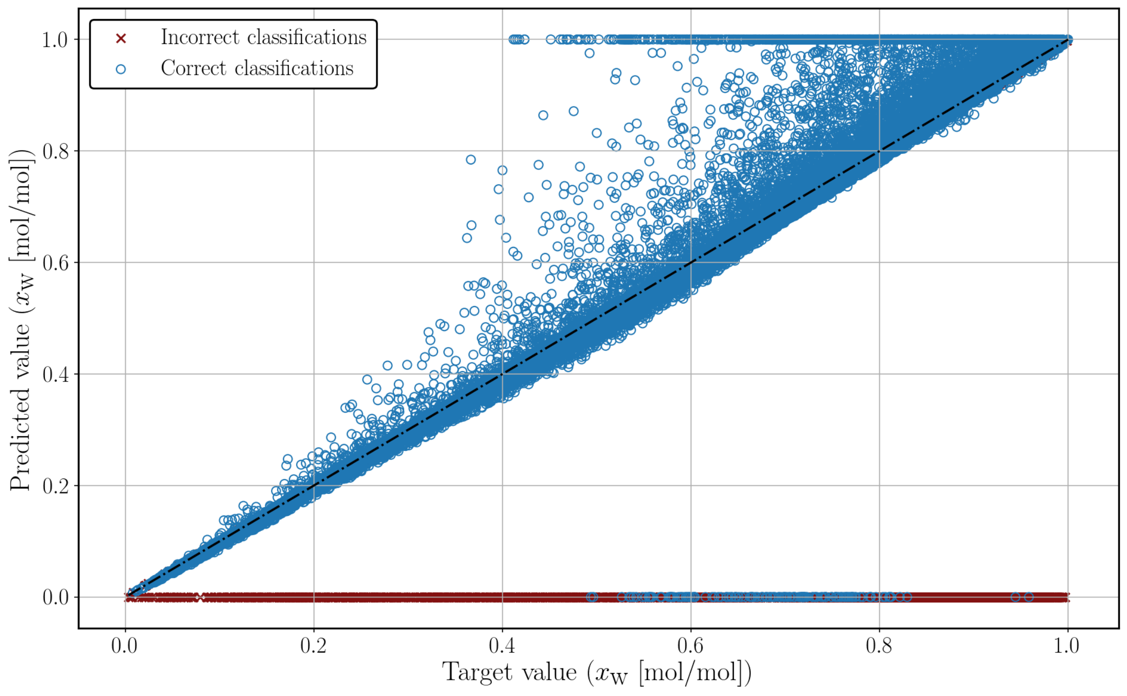


Figure I.21: Entropy-volume flash: two-phase liquid phase mole fraction (water) prediction correlation plot.

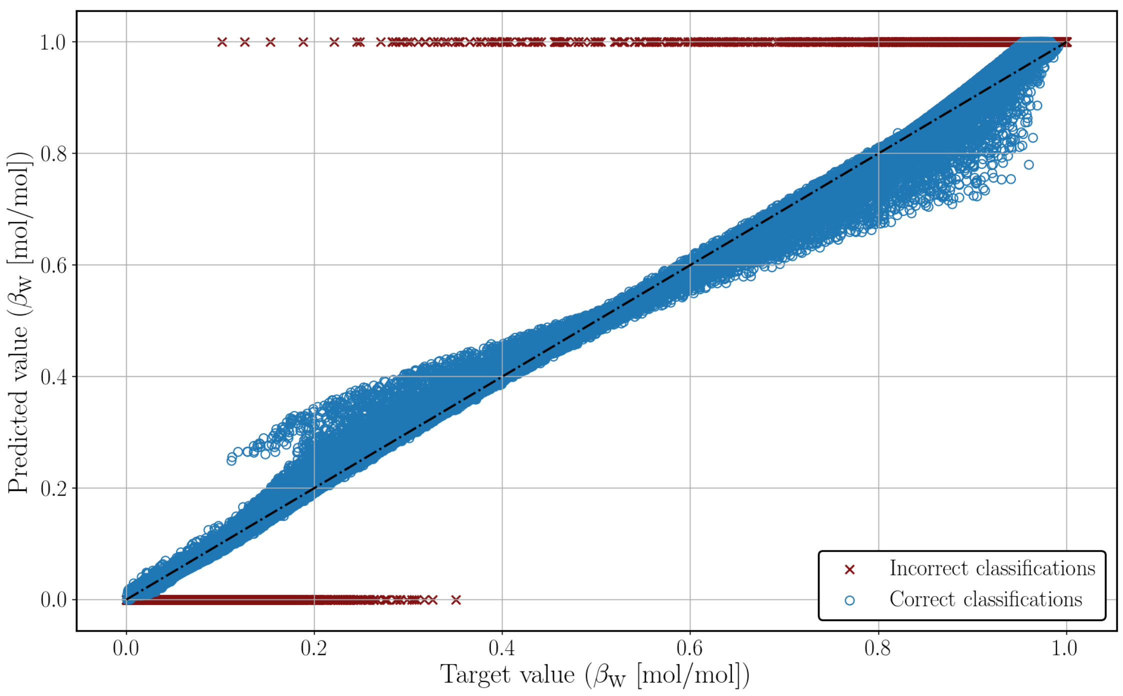


Figure I.22: Entropy-volume flash: two-phase water vapor fraction prediction correlation plot.

I.4 HV-flash regression network correlation plots

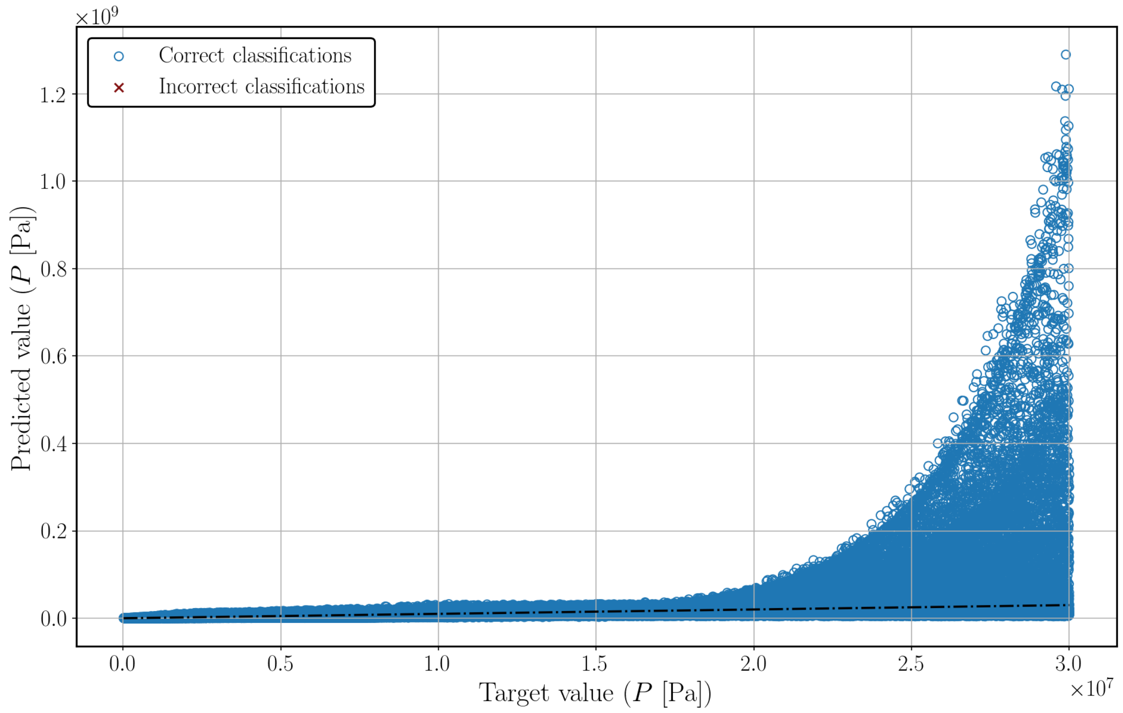


Figure I.23: Enthalpy-volume flash: liquid pressure prediction correlation plot.

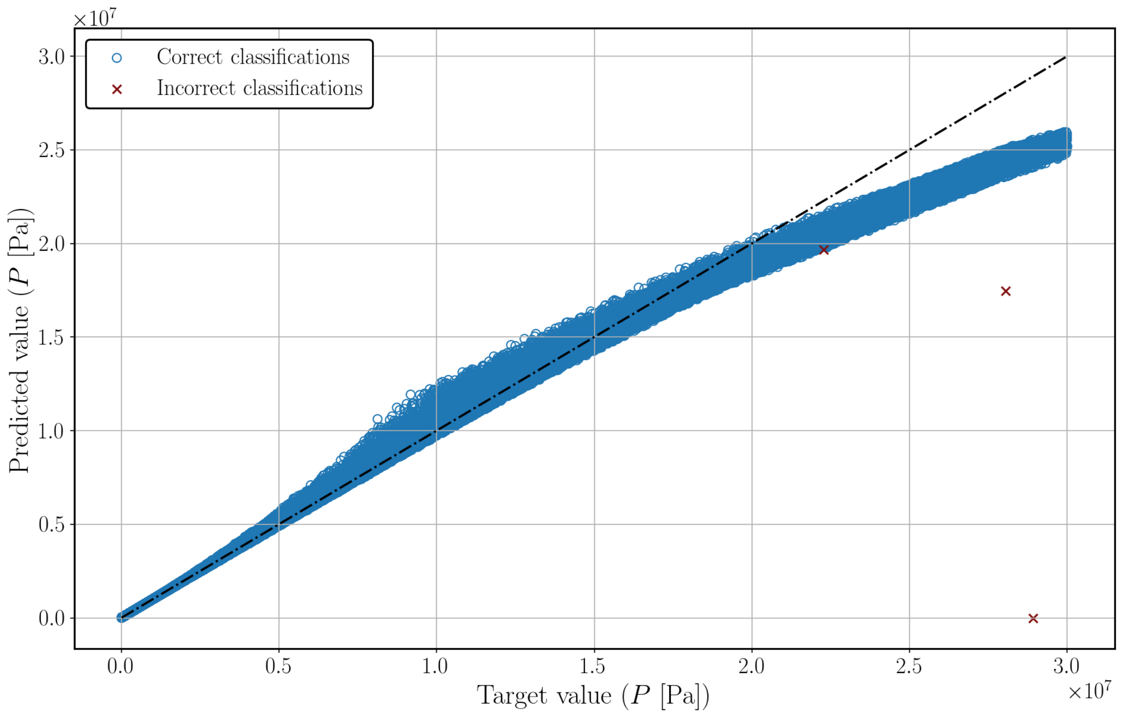


Figure I.24: Enthalpy-volume flash: vapor pressure prediction correlation plot.

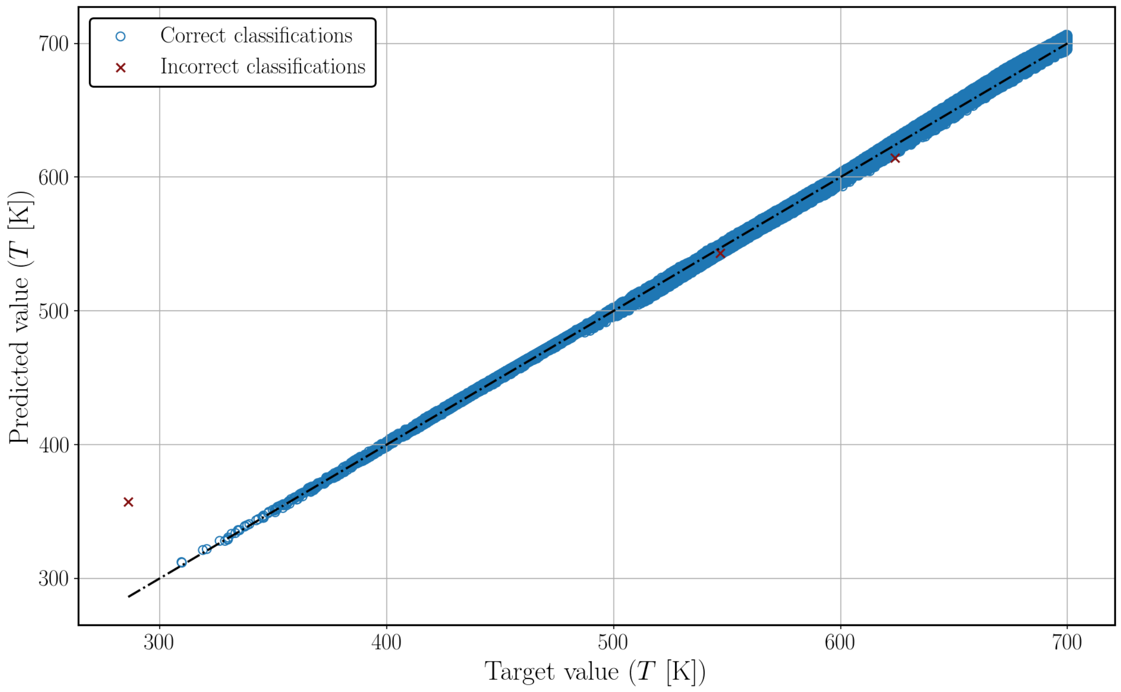


Figure I.25: Enthalpy-volume flash: vapor temperature prediction correlation plot.

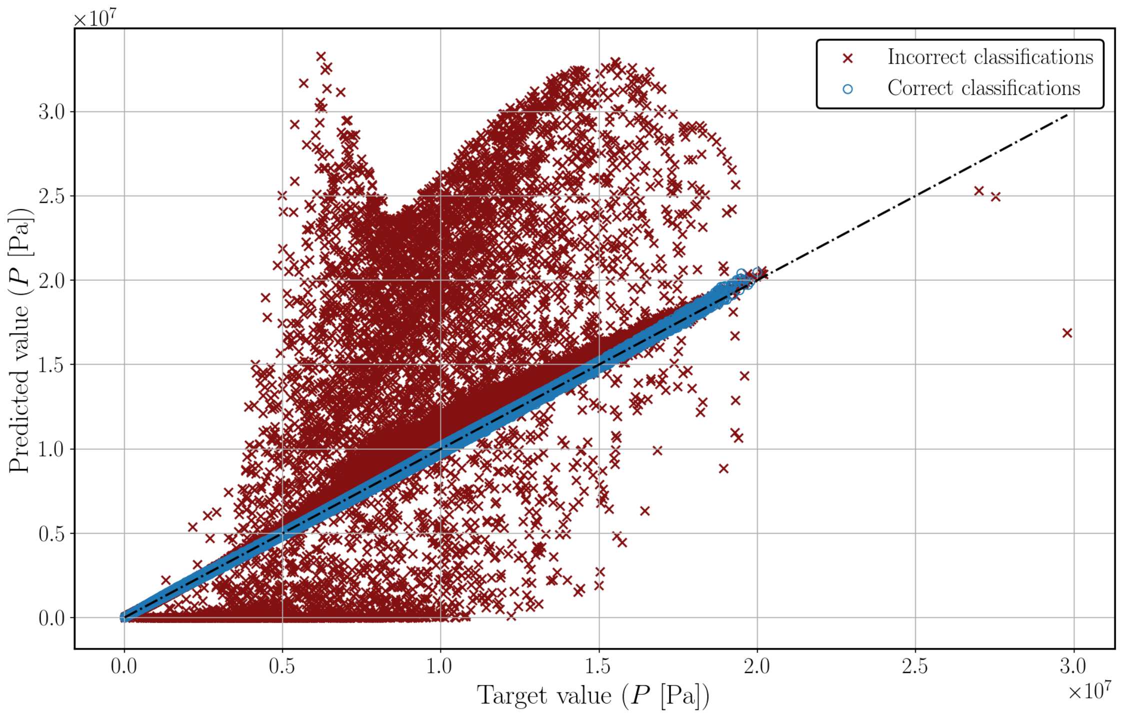


Figure I.26: Enthalpy-volume flash: two-phase pressure prediction correlation plot.

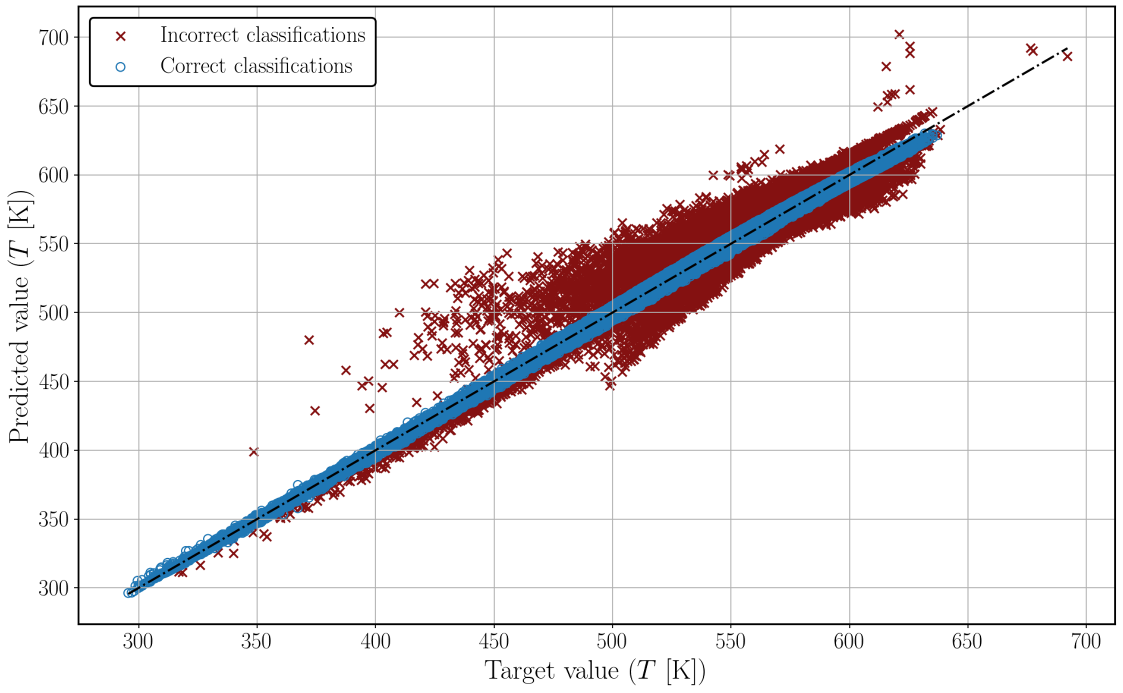


Figure I.27: Enthalpy-volume flash: two-phase temperature prediction correlation plot.

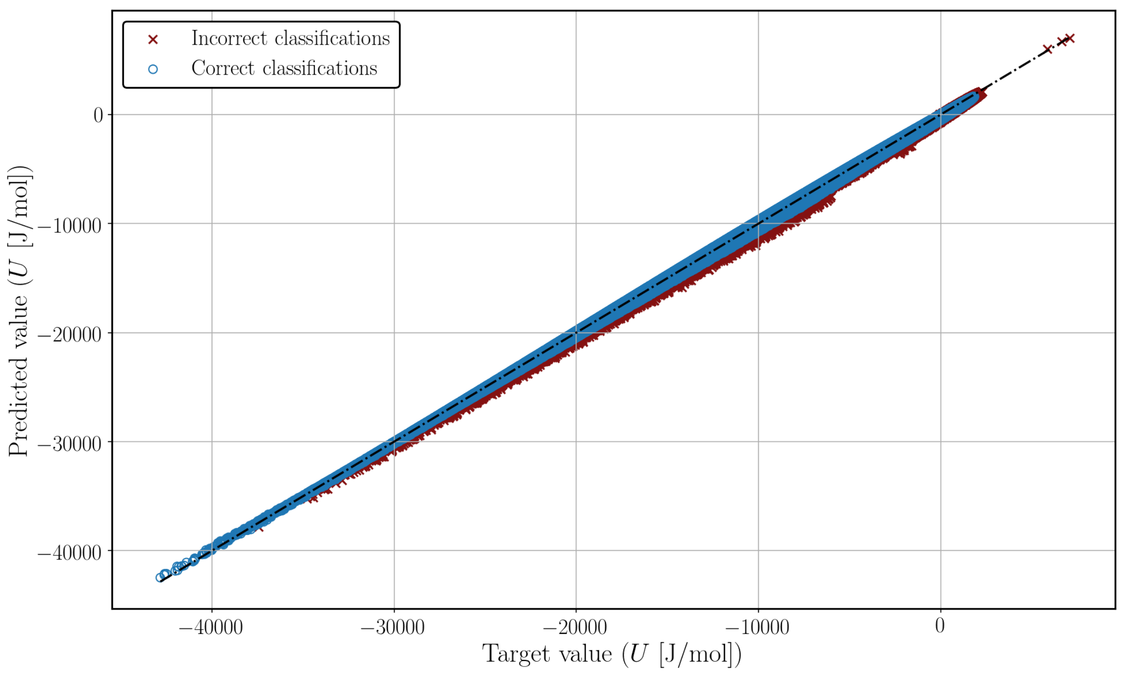


Figure I.28: Enthalpy-volume flash: two-phase internal energy prediction correlation plot.

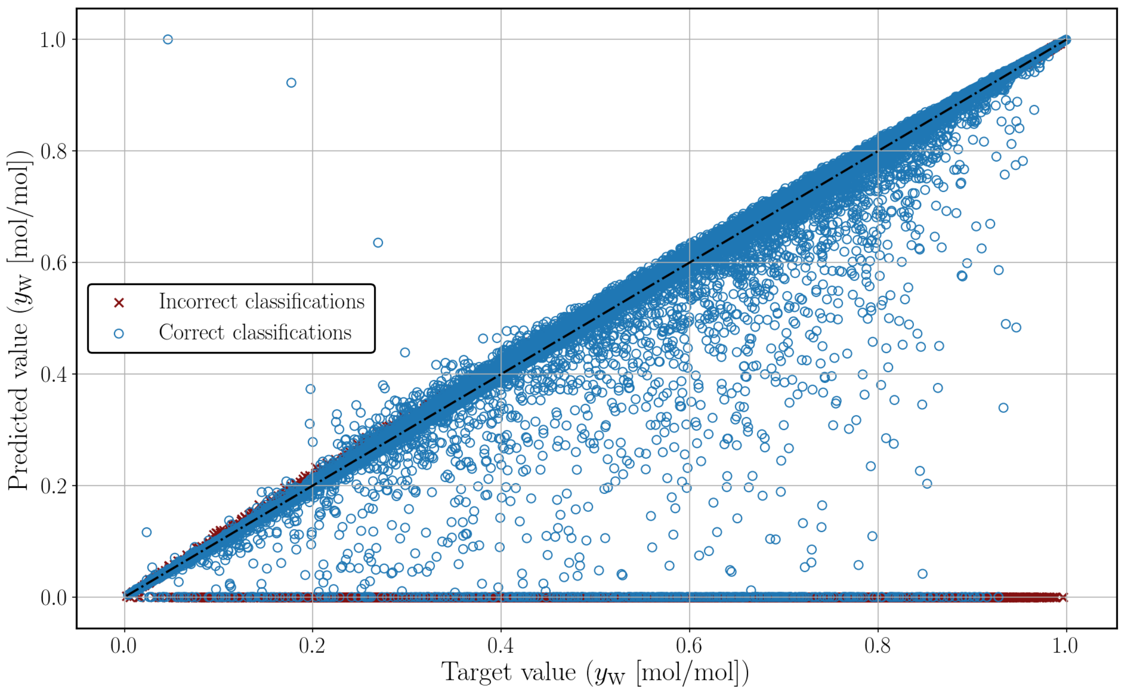


Figure I.29: Enthalpy-volume flash: two-phase vapor phase mole fraction (water) prediction correlation plot.

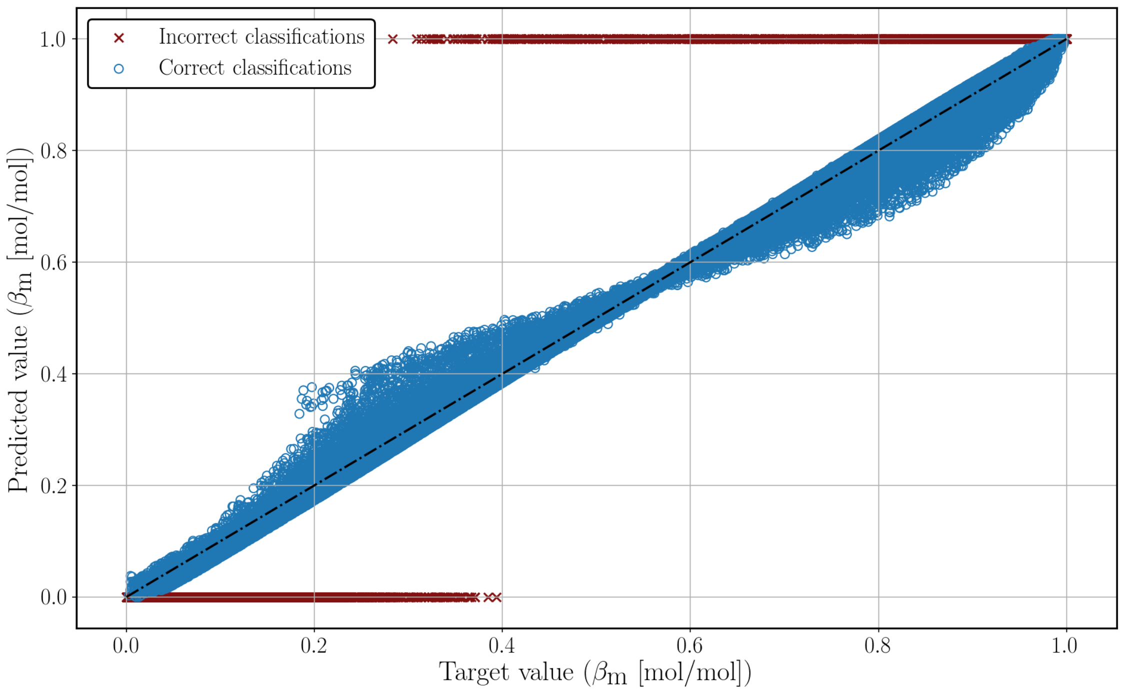


Figure I.30: Enthalpy-volume flash: two-phase methanol vapor fraction prediction correlation plot.

Appendix J

Additional recommendations

In chapter 6 the most important recommendations for future work were given. In this appendix, additional somewhat less important but still significant recommendations are provided.

J.1 Speed improvements

Optimizing current code

Code optimization was not considered part of the scope of this thesis, where possible the fastest known alternative was implemented, but there will likely be approaches not considered by the author that may lead to shorter execution times. Future research could go through the code to find and replace these parts of the source code.

Combining property predictions

Currently, each property has a single neural network that is used to predict its value, this was done in order to have more flexibility in what to predict, and increase computational efficiency if only one type of property is required. The downside of this approach is that if many properties are required at once, multiple networks will have to be executed, increasing computation times.

If certain property combinations are frequently required together, it might be interesting to train a single neural network on both properties at once to potentially save on time that would otherwise have to be spent on executing two different networks. While an improvement will likely be achievable through this approach, the actual speed increase will depend on the complexity of the network required, which will likely be higher for multiple outputs than for a single output.

J.2 Accuracy improvements

Currently no optimization efforts were undertaken to find the neural network settings (number of layers/neurons, activation function, loss function, training algorithm, etc.) that would lead to the highest predictive accuracies. Future work could look into using, for instance, genetic algorithms (GA) to find the optimal network topology for each property.

J.3 Stability

Higher predictive accuracy would most likely result in a more stable system of flashes.

In addition, if somehow a method can be developed which relates the different flashes to each other, such as what is done through the use of a common equation of state in conventional methods, stability could be improved, a hybrid algorihtm that combines neural networks with conventional methods as developed in [5] could be an interesting starting point.

J.4 Other recommendations

Specific data generation

In this thesis, data was generated using a grid-based approach, the upper and lower bounds of the property ranges of interest as well as the number of samples to generate for each range were specified, and data was generated by creating a grid within these bounds of evenly or logarithmically spaced points.

The main benefit of this approach was its simplicity, however, there are two main disadvantages to this approach:

- As the pressure, temperature two-phase region is rather narrow, the grid-based approach generated much fewer data points in the two-phase region than in the pure liquid and vapor regions. Thus, less data was available for training the two-phase networks, often leading to lower accuracies.
- In certain regions where property values would be relatively straight-forward to predict a lot of data was generated, while in other parts of the range of interest, where property values varied more, less data was available.

It could be interesting to specifically generate data in parts of the range of interest where the output data shows a large variance, and less where it is more or less linear or nearly constant. In this way, more data can be generated there where a higher detail is necessary, and less where less detail is required. For example see Figure J.1 which shows a linear and non-linear type of data generation: the curve of interest varies more in the middle section, and is nearly constant to the left and right of the bell-shape, thus it would be more useful to have less points there where the curve is flat (easy to interpolate between), and more data points where it varies.

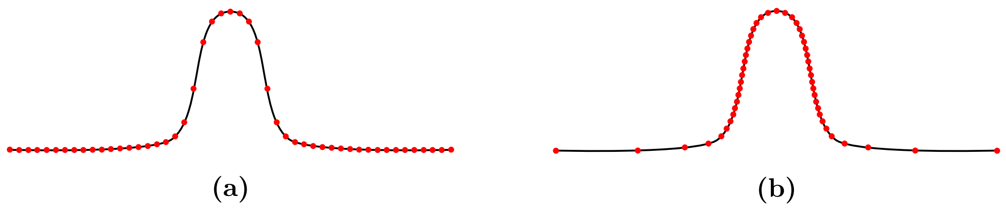


Figure J.1: Graphical example of a non-linear data generation scheme. In the first case, data was generated using a linear spacing (a), while in the second case, data was spaced based on the shape of the data curve of interest, leading to more data points where the curve varies more (b).

Generate and discard data

The amounts of data required to train a neural network on a binary mixture are already quite large. If higher accuracies are required, even more data will be required. Furthermore, if more complex mixtures are of interest the amount of data required will likely increase exponentially with the number of components. In this case not only data generation times will become a problem, but data storage might also become difficult. As an example, the project at hand already works with a few GB of data.

For this reason it might be interesting to look at the development of an algorithm that generates smaller parts of the data range of interest, partially trains a network on this data, and immediately discards it. In this way only a (relatively) small number of data instances have to be physically stored simultaneously.

Learning boundaries instead of entire classes

The current methods developed here use a classification algorithm to distinguish between phase regions. The network is trained on data from the entire range of interest, even though the only relevant parts for phase classification are the boundaries between the phases.

It could be interesting to look into training a network on the phase boundaries only, and using a kind of point-in-polygon algorithm to determine whether an input combination belongs to a given phase region. By training on the boundary only, less data is required than the current approach requires.

Additional property networks

In the current framework some properties (heat capacity, viscosity, and thermal conductivity) were calculated based on temperature correlations. In the future these temperature correlations could be replaced using neural network fits, although this was not considered of interest for this project.

Bibliography

[1] J. Seader, E. J. Henley, and D. K. Roper, *Separation Process Principles.* Hoboken: John Wiley & Sons, fourth ed., 2016.

[2] G. M. Kontogeorgis and R. Gani, *Computer-Aided Property Estimation for Process and Product Design.* Amsterdam: Elsevier B.V., first ed., 2004.

[3] M. L. Michelsen and J. M. Mollerup, *Thermodynamic Models: Fundamental & Computational Aspects.* Holte: Tie-Line Publications, second ed., 2007.

[4] V. Gaganis and N. Varotsis, "Machine Learning Methods to Speed up Compositional Reservoir Simulation," in *74th EAGE Conference & Exhibition incorporating SPE EUROPEC 2012*, no. June 2012, (Copenhagen, Denmark), pp. 4–7, Society of Petroleum Engineers, 2012.

[5] A. Kashinath, M. Szulczewski, and A. Dogru, "A fast algorithm for calculating isothermal phase behavior using machine learning," *Fluid Phase Equilibria*, vol. 465, pp. 73–82, 2018.

[6] AspenTech Documentation Team, "Aspen HYSYS Dynamic Modeling Guide," tech. rep., 2006.

[7] M. L. Michelsen, "Phase Equilibrium Calculations. What is Easy and What is Difficult?," *Computers & Chemical Engineering*, vol. 17, no. 5/6, pp. 431–439, 1993.

[8] M. Castier, R. Kanes, and L. N. Véchot, "Flash calculations with specified entropy and stagnation enthalpy," *Fluid Phase Equilibria*, vol. 408, pp. 196–204, 2016.

[9] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow.* O'Reilly Media, first ed., 2017.

[10] "Zero Emission Fuels - Zero emission methanol from sunlight and air.." http://www.zeroemissionfuels.com/ (last accessed 16 October 2018).

[11] "COCO Help: TEA." https://www.cocosimulator.org/index_help.php?page=TEA/tea.htm (last accessed 4 October 2018).

[12] "COCO - the CAPE-OPEN to CAPE-OPEN simulator." https://www.cocosimulator.org/index.html (last accessed 06 September 2018).

[13] R. Hołyst and A. Poniewierski, *Thermodynamics for Chemists, Physicists and Engineers.* Springer, first ed., 2012.

[14] J. Smith, H. Van Ness, and M. Abbott, *Introduction to Chemical Engineering Thermodynamics.* New York: McGraw-Hill Higher Education, sixth ed., 2003.

[15] S. M. Walas, *Phase Equilibria in Chemical Engineering.* Boston: Butterworth Publishers, first ed., 1985.

[16] R. A. Alberty, "Use of Legendre Transforms in Chemical Thermodynamics," *Pure Applied Chemistry*, vol. 73, no. 8, pp. 1349–1380, 2001.

[17] I. Mills, T. Cvitas, K. Homann, N. Kallay, and K. Kuchitsu, *Quantities, Units and Symbols in Physical Chemistry.* Blackwell Science, second ed., 1993.

[18] J. D. van der Waals, "The equation of state for gases and liquids," in *Nobel Lectures, Physics 1901-1921*, vol. i, pp. 254–265, Amsterdam: Elsevier Publishing Company, 1967.

[19] D. Y. Peng and D. B. Robinson, "A New Two-Constant Equation of State," *Industrial and Engineering Chemistry Fundamentals*, vol. 15, no. 1, pp. 59–64, 1976.

[20] G. Soave, "Equilibrium constants from a modified Redlich-Kwong equation of state," *Chemical Engineering Science*, vol. 27, no. 6, pp. 1197–1203, 1972.

[21] C. H. Whitson and M. L. Michelsen, "The negative flash," *Fluid Phase Equilibria*, vol. 53, pp. 51–71, 1989.

[22] M. L. Michelsen, "The Isothermal Flash Problem. Part II. Phase-Split Calculation," *Fluid Phase Equilibria*, vol. 9, pp. 21–40, 1982.

[23] M. L. Michelsen, "Multiphase Isenthalpic and Isentropic Algorithms," *Fluid Phase Equilibria*, vol. 33, pp. 13–27, 1987.

[24] T. Jindrová and J. Mikyška, "Fast and robust algorithm for calculation of two-phase equilibria at given volume, temperature, and moles," *Fluid Phase Equilibria*, vol. 353, pp. 101–114, 2013.

[25] M. Castier, "Solution of the isochoric-isoenergetic flash problem by direct entropy maximization," *Fluid Phase Equilibria*, vol. 276, no. 1, pp. 7–17, 2009.

[26] M. L. Michelsen, "State function based flash specifications," *Fluid Phase Equilibria*, vol. 158, pp. 617–626, 1999.

[27] M. Petitfrere and D. V. Nichita, "Robust and efficient Trust-Region based stability analysis and multiphase flash calculations," *Fluid Phase Equilibria*, vol. 362, pp. 51–68, 2014.

[28] H. Fatoorehchi, H. Abolghasemi, and R. Rach, "A new parametric algorithm for isothermal fl ash calculations by the Adomian decomposition of Michaelis – Menten type nonlinearities," *Fluid Phase Equilibria*, vol. 395, pp. 44–50, 2015.

[29] M. L. Michelsen, "The isothermal flash problem. Part I. Stability," *Fluid Phase Equilibria*, vol. 9, no. 1, pp. 1–19, 1982.

[30] J. F. Boston and H. I. Britt, "A radically different formulation and solution of the single-stage flash problem," *Computers and Chemical Engineering*, vol. 2, no. 2-3, pp. 109–122, 1978.

[31] V. S. Parekh and P. M. Mathias, "Efficient Flash Calculations for Chemical Process Design — Extension of the Boston–Britt 'Inside–out' Flash Algorithm to Extreme Conditions and New Flash Types," *Computers and Chemical Engineering*, vol. 22, no. 10, pp. 1371–1380, 1998.

[32] Y. Li, R. T. Johns, and K. Ahmadi, "A rapid and robust alternative to Rachford-Rice in flash calculations," *Fluid Phase Equilibria*, vol. 316, pp. 85–97, 2012.

[33] C. Rasmussen, K. Krejbjerg, M. Michelsen, and K. Bjurstrøm, "Increasing the Computational Speed of Flash Calculations With Applications for Compositional, Transient Simulations," *SPE Reservoir Evaluation & Engineering*, vol. 9, pp. 5–8, 2006.

[34] J. Mikyška and A. Firoozabadi, "Investigation of mixture stability at given volume, temperature, and number of moles," *Fluid Phase Equilibria*, vol. 321, pp. 1–9, 2012.

[35] M. Castier, "Helmholtz function-based global phase stability test and its link to the isothermal-isochoric flash problem," *Fluid Phase Equilibria*, vol. 379, pp. 104–111, 2014.

[36] H. Gecegormez and Y. Demirel, "Phase stability analysis using interval Newton method with NRTL model," *Fluid Phase Equilibria*, vol. 237, no. 1-2, pp. 48–58, 2005.

[37] D. V. Nichita, "Fast and robust phase stability testing at isothermal-isochoric conditions," *Fluid Phase Equilibria*, vol. 447, pp. 107–124, 2017.

[38] T. Smejkal and J. Mikyška, "Phase stability testing and phase equilibrium calculation at specified internal energy, volume, and moles," *Fluid Phase Equilibria*, vol. 431, pp. 61–88, 2017.

[39] T. Smejkal and J. Mikyška, "Unified presentation and comparison of various formulations of the phase stability and phase equilibrium calculation problems," *Fluid Phase Equilibria*, vol. 476, pp. 61–88, 2018.

[40] W. Sun and Y.-X. Yuan, *Optimization Theory and Methods.* Boston: Science, first ed., 2006.

[41] D. V. Nichita, D. Broseta, and F. Montel, "Calculation of convergence pressure/temperature and stability test limit loci of mixtures with cubic equations of state," *Fluid Phase Equilibria*, vol. 261, no. 1-2, pp. 176–184, 2007.

[42] M. Seifi and J. Abedi, "An efficient and robust saturation pressure calculation algorithm for petroleum reservoir fluids using a neural network," *Petroleum Science and Technology*, vol. 30, no. 22, pp. 2329–2340, 2012.

[43] A. I. Galushkin, *Neural Networks Theory.* Berlin; Heidelberg: Springer-Verlag, first ed., 2007.

[44] R. J. Schalkhoff, *Artificial Neural Networks.* McGraw-Hill, first ed., 1997.

[45] B. Macukow, "Neural Networks – State of Art, Brief History, Basic Models and Architecture," in *IFIP International Federation for Information Processing*, vol. 9842, pp. 3–14, Springer International Publishing Switzerland, 2016.

[46] A. Statnikov, C. F. Aliferis, I. Tsamardinos, D. Hardin, and S. Levy, "A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis," *Bioinformatics*, vol. 21, no. 5, pp. 631–643, 2005.

[47] E. Guresen, G. Kayakutlu, and T. U. Daim, "Using artificial neural network models in stock market index prediction," *Expert Systems with Applications*, vol. 38, no. 8, pp. 10389–10397, 2011.

[48] T. Sagara and M. Hagiwara, "Natural language neural network and its application to question-answering system," *Neurocomputing*, vol. 142, pp. 201–208, 2014.

[49] Y. Zhao, X. Du, G. Xia, and L. Wu, "A novel algorithm for wavelet neural networks with application to enhanced PID controller design," *Neurocomputing*, vol. 158, pp. 257–267, 2015.

[50] U. Fiore, A. De Santis, F. Perla, P. Zanetti, and F. Palmieri, "Using generative adversarial networks for improving classification effectiveness in credit card fraud detection," *Information Sciences*, vol. 0, pp. 1–8, 2017.

[51] J. Dou, C. Liu, and B. Wang, "Short-term Wind Power Forecasting Based on Convolutional Neural Networks," *IOP Conf. Series: Earth and Environmental Science*, vol. 170, no. 4, p. 042023, 2018.

[52] H. Li, P. Wang, M. You, and C. Shen, "Reading car license plates using deep neural networks," *Image and Vision Computing*, vol. 72, pp. 14–23, 2018.

[53] R. Rojas, *Neural Networks: A Systematic Introduction.* Berlin; Heidelberg: Springer-Verlag, first ed., 1996.

[54] J. Schmidhuber, "Deep Learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[55] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 9, (Chia, Sardinia, Italy), pp. 249–256, 2010.

[56] A.-D. Almási, S. Woźniak, V. Cristea, Y. Leblebici, and T. Engbersen, "Review of advances in neural networks: Neural design technology stack," *Neurocomputing*, vol. 174, pp. 31–41, 2016.

[57] A. Prieto, B. Prieto, E. M. Ortigosa, E. Ros, F. Pelayo, J. Ortega, and I. Rojas, "Neural networks: An overview of early research, current frameworks and new challenges," *Neurocomputing*, vol. 214, pp. 242–268, 2016.

[58] D. A. Sprecher, "On the Structure of Continuous Functions of Several Variables," *Transactions of the American Mathematical Society*, vol. 115, pp. 340–355, 1964.

[59] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, Dec 1989.

[60] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[61] S. Sonoda and N. Murata, "Neural network with unbounded activation functions is universal approximator," *Applied and Computational Harmonic Analysis*, vol. 43, no. 2, pp. 233–268, 2017.

[62] T. Yato and T. Seta, "Complexity and Completeness of Finding Another Solution and Its Application to Puzzles," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E86-A, no. 5, pp. 1052–1060, 2003.

[63] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017.

[64] "Keras Documentation." `https://keras.io/` (last accessed 09 September 2018).

[65] K. Janocha and W. M. Czarnecki, "On Loss Functions for Deep Neural Networks in Classification," *Schedae Informaticae*, vol. 25, pp. 49–59, 2016.

[66] A. Gepperth and B. Hammer, "Incremental learning algorithms and applications," in *European Symposium on Artificial Neural Networks ({ESANN})*, no. April, pp. 357–368, 2016.

[67] A. Antoniou and W. Lu, *Practical Optimization - Algorithms and Engineering Applications.* US: Springer, first ed., 2007.

[68] J. W. Chinneck, *Practical Optimization: a Gentle Introduction.* 2015.

[69] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016.

[70] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[71] J. E. Schmitz, R. J. Zemp, and M. J. Mendes, "Artificial neural networks for the solution of the phase stability problem," *Fluid Phase Equilibria*, vol. 245, no. 1, pp. 83–87, 2006.

[72] V. Gaganis and N. Varotsis, "Non-iterative phase stability calculations for process simulation using discriminating functions," *Fluid Phase Equilibria*, vol. 314, pp. 69–77, 2012.

[73] A. Farzi and A. T. Nejad, "Prediction of phase equilibria in binary systems containing acetone using artificial neural network," *International Journal of Scientific & Engineering Research*, vol. 6, no. 9, pp. 5–10, 2015.

[74] B. Vaferi, Y. Rahnama, P. Darvishi, A. Toorani, and M. Lashkarbolooki, "Phase equilibria modeling of binary systems containing ethanol using optimal feedforward neural network," *The Journal of Supercritical Fluids*, vol. 84, pp. 80–88, 2013.

[75] M. Moghadam and S. Asgharzadeh, "On the application of artificial neural network for modeling liquid-liquid equilibrium," *Journal of Molecular Liquids*, vol. 220, pp. 339–345, 2016.

[76] A. G. Ivakhnenko, "The review of problems solvable by algorithms of the group method of data handling (GMDH)," *Pattern Recognition and Image Analysis/Raspoznavaniye Obrazov I Analiz Izobrazhenii*, vol. 5, pp. 527–535, 1995.

[77] M. C. Iliuta, I. Iliuta, and F. Larachi, “Vapour-liquid equilibrium data analysis for mixed solvent-electrolyte systems using neural network models,” *Chemical Engineering Science*, vol. 55, no. 15, pp. 2813–2825, 2000.

[78] S. Mohanty, “Estimation of vapour liquid equilibria of binary systems, carbon dioxide-ethyl caproate, ethyl caprylate and ethyl caprate using artificial neural networks,” *Fluid Phase Equilibria*, vol. 235, no. 1, pp. 92–98, 2005.

[79] A. Ghaemi, S. Shahhoseini, M. G. Marageh, and M. Farrokhi, “Prediction of vapor-liquid equilibrium for aqueous solutions of electrolytes using artificial neural networks,” *Journal of Applied Sciences*, vol. 8, no. 4, pp. 615–621, 2008.

[80] M. Lashkarbolooki, Z. S. Shafipour, A. Z. Hezave, and H. Farmani, “Use of artificial neural networks for prediction of phase equilibria in the binary system containing carbon dioxide,” *Journal of Supercritical Fluids*, vol. 75, pp. 144–151, 2013.

[81] A. Chouai, S. Laugier, and D. Richon, “Modeling of thermodynamic properties using neural networks,” *Fluid Phase Equilibria*, vol. 199, no. 1-2, pp. 53–62, 2002.

[82] E. R. Mora, P. Carlos, F. Gonza, J. D. Dios, and D. Ocampo, “Thermodynamic properties of refrigerants using artificial neural networks,” *International Journal of Refrigeration*, vol. 6, pp. 9–16, 2014.

[83] J. O. Valderrama, A. Rea´tegui, and R. E. Rojas, “Density of Ionic Liquids Using Group Contribution and Artificial Neural Networks,” *Industrial & Engineering Chemistry Research*, vol. 48, no. 6, pp. 3254–3259, 2009.

[84] J. O. Valderrama, C. A. Fau, and V. J. Vicencio, “Artificial Neural Networks and the Melting Temperature of Ionic Liquids,” *Industrial & Engineering Chemistry Research*, vol. 53, pp. 10504–10511, 2014.

[85] J. Homer, S. C. Generalis, and J. H. Robson, “Artificial neural networks for the prediction of liquid viscosity, density, heat of vaporization, boiling point and Pitzer’s acentric factor. Part I. Hydrocarbons,” *Physical Chemistry Chemical Physics*, vol. 1, no. 17, pp. 4075–4081, 1999.

[86] R. Haghbakhsh, H. Adib, P. Keshavarz, M. Koolivand, and S. Keshtkari, “Development of an artificial neural network model for the prediction of hydrocarbon density at high-pressure, high-temperature conditions,” *Thermochimica Acta*, vol. 551, pp. 124–130, 2013.

[87] M. Moosavi and N. Soltani, “Prediction of hydrocarbon densities using an artificial neural network-group contribution method up to high temperatures and pressures,” *Thermochimica Acta*, vol. 556, pp. 89–96, 2013.

[88] S. Laugier and D. Richon, “Use of artificial neural networks for calculating derived thermodynamic quantities from volumetric property data,” *Fluid Phase Equilibria*, vol. 210, no. 2, pp. 247–255, 2003.

[89] A. Sözen, M. Özalp, and E. Arcaklioğlu, “Calculation for the thermodynamic properties of an alternative refrigerant (R508b) using artificial neural network,” *Applied Thermal Engineering*, vol. 27, no. 2-3, pp. 551–559, 2007.

[90] Á. Mulero, I. Cachadiña, and J. Valderrama, "Artificial neural network for the correlation and prediction of surface tension of refrigerants," *Fluid Phase Equilibria*, vol. 451, pp. 60–67, 2017.

[91] M. Al-Marhoun and E. Osman, "Using Artificial Neural Networks to Develop New PVT Correlations for Saudi Crude Oils," in *Abu Dhabi International Petroleum Exhibition and Conference*, Society of Petroleum Engineers, 2002.

[92] M. Hosseinzadeh and A. Hemmati-Sarapardeh, "Toward a predictive model for estimating viscosity of ternary mixtures containing ionic liquids," *Journal of Molecular Liquids*, vol. 200, no. PB, pp. 340–348, 2014.

[93] P. Díaz-Rodríguez, J. C. Cancilla, G. Matute, and J. S. Torrecilla, "Viscosity estimation of binary mixtures of ionic liquids through a multi-layer perceptron model," *Journal of Industrial and Engineering Chemistry*, vol. 21, pp. 1350–1353, 2015.

[94] J. Hekayati and M. R. Rahimpour, "Estimation of the saturation pressure of pure ionic liquids using MLP artificial neural networks and the revised isofugacity criterion," *Journal of Molecular Liquids*, vol. 230, pp. 85–95, 2017.

[95] A. A. Rohani, G. Pazuki, H. A. Najafabadi, S. Seyfi, and M. Vossoughi, "Comparison between the artificial neural network system and SAFT equation in obtaining vapor pressure and liquid density of pure alcohols," *Expert Systems with Applications*, vol. 38, no. 3, pp. 1738–1747, 2011.

[96] F. Gharagheizi, A. Eslamimanesh, B. Tirandazi, A. H. Mohammadi, and D. Richon, "Handling a very large data set for determination of surface tension of chemical compounds using Quantitative Structure-Property Relationship strategy," *Chemical Engineering Science*, vol. 66, no. 21, pp. 4991–5023, 2011.

[97] J. Taskinen and J. Yliruusi, "Prediction of physicochemical properties based on neural network modelling," *Advanced Drug Delivery Reviews*, vol. 55, no. 9, pp. 1163–1183, 2003.

[98] S. S. Sablani, O.-D. Baik, and M. Marcotte, "Neural networks for predicting thermal conductivity of bakery products," *Journal of Food Engineering*, vol. 52, no. 3, pp. 299–304, 2002.

[99] M. S. Rahman, M. M. Rashid, and M. A. Hussain, "Thermal conductivity prediction of foods by Neural Network and Fuzzy (ANFIS) modeling techniques," *Food and Bioproducts Processing*, vol. 90, no. 2, pp. 333–340, 2012.

[100] P. Carlès, "A brief review of the thermophysical properties of supercritical fluids," *Journal of Supercritical Fluids*, vol. 53, no. 1-3, pp. 2–11, 2010.

[101] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR 2015*, pp. 1–15, 2015.

[102] T. Dozat, "Incorporating Nesterov Momentum into Adam," Tech. Rep. 1, 2016.

[103] S. Makridakis, "Accuracy concerns measures : theoretical and practical concerns," *International journal of forecasting*, vol. 9, pp. 527–529, 1993.

[104] E. H. Benmekki, T. Kwak, and G. A. Mansoori, "Van der Waals Mixing Rules for Cubic Equations of State," in *Supercritical Fluids*, ch. 9, pp. 101–114, 1987.

[105] H. A. Kooijman and R. Taylor, *The ChemSep Book.* second ed., 2006.

[106] S. Kirkpatrick, C. D. J. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[107] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, 1995.

[108] H. R. Hansen, “Global Optimization Using Interval Analysis : The One-Dimensional Case,” *Journal of Optimization Theory and Applications*, vol. 29, no. 3, pp. 331–344, 1979.

[109] M. Wetter and J. Wright, “A comparison of deterministic and probabilistic optimization algorithms for nonsmooth simulation-based optimization,” *Building and Environment*, vol. 39, no. 8 SPEC. ISS., pp. 989–999, 2004.

[110] J. A. Zoll Weg and G. W. Mulholland, “On the law of the rectilinear diameter,” *The Journal of Chemical Physics*, vol. 57, no. 3, pp. 1021–1025, 1972.