March 2022

The Application of Physics Informed Neural Networks to Compositional Modeling

Thelma A. Ihunde
Louisiana State University and Agricultural and Mechanical College

THE APPLICATION OF PHYSICS INFORMED NEURAL NETWORKS TO COMPOSITIONAL MODELING

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science

in

The Department of Petroleum Engineering

by
Thelma Anizia Ihunde
B.S., Louisiana State University, 2017
M.B.A., Louisiana State University, 2019
May 2022

TABLE OF CONTENTS

ABSTRACT

Compositional modeling is essential when simulating processes involving significant changes in reservoir fluid composition. It is computationally expensive because we typically need to predict the states and properties of multicomponent fluid mixtures at several different points in space and time. To speed up this process, several researchers have used machine learning algorithms to train deep learning (DL) models on data from the rigorous phase-equilibrium (flash) calculations. However, one shortcoming of the DL models is that there is no explicit consideration for the governing physics. So, there is no guarantee that the model predictions will honor the thermodynamical constraints of phase equilibrium (Ihunde & Olorode, 2022).

This work is the first attempt to incorporate thermodynamics constraints into the training of DL models to ensure that they yield two-phase flash predictions that honor the physical laws that govern phase equilibrium. A space-filling mixture design is used to generate one million different compositions at different pressures (Ihunde & Olorode, 2022). Stability analysis and flash calculations are performed on these compositions to obtain the corresponding phase compositions and vapor fraction (Ihunde & Olorode, 2022). Physics-informed neural network (PINN) and standard deep neural network (DNN) models were trained to predict two-phase flash results using the data from the actual phase-equilibrium calculations (Ihunde & Olorode, 2022).

Considering the stochasticity of the deep learning optimization process, we used the seven-fold cross-validation to obtain reliable estimates of average model accuracy and variance (Ihunde & Olorode, 2022). Comparing the PINN and standard DNN models reveals that PINNs can incorporate physical constraints into DNNs without significantly lowering the model accuracy (Ihunde & Olorode, 2022). The evaluation of the model results with the test data shows that both

PINN and standard DNN models yield coefficients of determination of ~97% (Ihunde & Olorode, 2022). However, the root-mean-square error of the physics-constraint errors in the PINN model is over 55% lower than that of the standard DNN model (Ihunde & Olorode, 2022). This indicates that PINNs significantly outperform DNNs in honoring the governing physics. Finally, we demonstrate the significance of honoring the governing physics by comparing the resulting phase envelopes obtained from overall compositions computed from the PINN, DNN, and linear regression model predictions (Ihunde & Olorode, 2022).

1. INTRODUCTION

1.1. Background

Compositional fluid models describe the complex phase behavior of reservoir fluids. Two-phase compositional modeling typically involves stability analysis to determine the preferred stable state for the fluid mixture, after which the Rachford-Rice equation is solved iteratively to obtain the vapor fraction (Rachford & Rice, 1952). Each component's mole fraction in each phase is also computed in this iterative two-phase flash procedure. To obtain the initial guess for these mole fractions, we typically estimate the vapor-liquid equilibrium factor (K-factor) using Wilson's correlation (Wilson, 1968). While simulating compositional fluid flow in porous media (Coats, 1980; Young & Stephenson, 1983; Pal & Mandal, 2021) and pipes (Gould, 1979; Furukawa et al., 1986), phase equilibrium calculations are required at every gridblock and time step.

Modeling the fluid property changes in space and time is very computationally expensive. To speed up the computationally demanding phase equilibrium computations in compositional modeling, numerous researchers have developed iterative and non-iterative methodologies.. Some of the iterative techniques include the minimization of Gibbs free energy (Nichita et al., 2002), dimensionality reduction (Firoozabadi & Pan, 2000; Pan & Firoozabadi, 2001; Nichita & Graciaa, 2011), and solving the Rachford-Rice (RR) equation by minimizing a non-monotonic convex function (Okuno et al., 2010). Non-iterative approaches include linear interpolation based on look-up tables. (Voskov & Tchelepi, 2009; Belkadi et al., 2011; Wu et al., 2015).

In recent years, researchers have done a lot of work to improve the speed of flash calculations involving approaches utilizing Machine Learning (ML). They have demonstrated that machine learning algorithms can improve the speed and stability (convergence) of compositional simulation

of unconventional reservoirs by at least 90% while maintaining very high accuracy (Gaganis & Varotsis, 2012, 2014; Kashinath et al., 2018;K. Wang et al., 2019; Wang et al., 2020; Li et al., 2019; S. Wang et al., 2019; Wang et al., 2020).

The application of ML to stability analysis and flash calculations is primarily through supervised learning. The ML algorithm learns a target function that best maps the training data input during this supervised learning process. Based on the curvature of fluid phase envelopes, this target function is expected to be highly non-linear to effectively capture the pressure, temperature, and composition relationship. The trained machine learning model is then used to make predictions on any new dataset. Since ML algorithms focus on minimizing the model error during training and not on the functional form, non-parametric supervised ML models have been mostly used for compositional modeling. These include Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), and Relevance Vector Machines (RVMs).

1.2. Motivation

According to research, trained machine learning models can predict flash calculation outcomes two orders of magnitude faster than the traditional iterative flash method (Li et al., 2019). Furthermore, the flash calculation results from DL models yield an excellent match when implemented into compositional reservoir simulators in place of the standard flash procedure (Wang et al., 2020). However, DL models can yield predictions that minimize the loss function but do not honor the underlying physics of phase equilibrium. This is because, in the DL models, the loss function is simply the disparity between the training data and the model prediction of phase mole fractions (x_i and y_i) and vapor fractions (V) (Ihunde & Olorode, 2022). These standard DL models typically yield model predictions of x_i, y_i, and V that closely match the training data,

but are unphysical because the combination of these predictions according to the interphase mass balance equation (given in Section 3.2) shows considerable errors.

Numerous authors have built frameworks that enable the incorporation of physics into the training of DNN models to solve the challenge of honoring physics during the training of DL models. The most popular method to use involves adding initial and boundary conditions of governing partial differential equations (PDEs) as penalties in the loss function (Raissi & Karniadakis, 2018; Raissi et al., 2019; and Haghighat & Juanes, 2021). Other researchers have also demonstrated the potential of implementing PDE constraints as extensions to the computational graph for the deep neural network (Huang et al., 2020; Xu & Darve, 2020). However, no research has been published on honoring physics when training deep learning models on data from two-phase flash. An initial attempt to address this limitation is published in Ihunde & Olorode, 2022

This work aims to evaluate the feasibility of using the penalty approach to impose thermodynamics constraints such as interphase mass balance and component balance during the training of deep neural network models. Throughout this work, I will refer to this approach as the Physics Informed Neural Network (PINN), but it could also be referred to as a physics-constrained deep learning (PCDL) model. This work also evaluates the use of classification accuracy as a performance measure for artificial neural networks trained on data for phase identification.

This thesis is organized as follows—Chapter 2 reviews previously published work on machine learning approaches for compositional modeling, physics-informed neural networks, and experimental design for mixtures. Chapter 3 discusses the two-phase flash calculation process with particular emphasis on the thermodynamic constraints. Chapter 4 discusses deep neural networks and how standard deep neural networks are modified to incorporate physics constraints. It also

discusses the model evaluation metrics used to quantify the extent to which the predictions made by the standard DNN and PINN models honor thermodynamic constraints. Chapter 5 describes the methodology for the experimental design to create the compositions for the reservoir fluids that are employed in the data generation process to train the deep learning models. This chapter also discusses how the deep learning models are implemented. Chapter 6 presents the results of implementing the PINN model and compares its performance to those of the DNN and linear regression models. Finally, chapter 7 summarizes and concludes the work done.

2. LITERATURE REVIEW

The process of compositional modeling involves performing flash calculations to obtain the mole fractions of each hydrocarbon component in each phase, given the pressure, temperature, and overall mole fraction. The standard approach for two-phase vapor-liquid equilibrium calculations requires that the chemical potential (or fugacity) of each component in the vapor phase is equal to that of the corresponding component in the liquid phase. It also requires that the phase mole fractions and overall mole fractions sum up to one and that the amount of each component is conserved across the phases. These two constraints are the component material balance and interphase mass balance constraints, respectively.

Researchers have suggested several approaches to accelerate compositional modeling with machine learning. Some of these involve replacing the stability test and flash calculations with proxy models developed using ML algorithms (K. Wang et al., 2019; S. Wang et al., 2019). In contrast, other methodologies use machine learning models to make vapor-liquid equilibrium coefficient (K- value) predictions that initialize the flash calculation instead of using Wilson's method (Gaganis & Varotsis, 2012; Kashinath et al., 2018).

Gaganis & Varotsis, 2012 created proxy models, using two ML models—the Support Vector Classifier (SVC) in the form of a discriminating function for the phase stability identification and a single layer ANN to predict the output results from two-phase flash calculations. The overall mole fraction, pressure, and temperature are used as the ANN inputs for the phase equilibrium problem to obtain the mixture components' K-values. SVC and ANN models were then used to make the predictions for any given overall mole fraction, pressure, and temperature. The results from their work showed that CPU time per phase calculation for an eight-component mixture

reduced by 92% in a conventional simulation run compared to their boosted simulator. They also showed that the CPU time increases as the number of components in the reservoir fluid increases.

Gaganis & Varotsis, 2014 demonstrated an integrated approach where stability analysis and flash calculations were directly predicted using classification and regression models. The Support vector machines (SVM) were used for stability analysis/phase classification, and ANN regression models were used to determine the phase split reduced variables from which the equilibrium phases properties and molar fractions are determined. The simulation results for an eight-component reservoir fluid showed the computation time per flash calculation improved by a speed-up factor of 27 from conventional iterative methods for stability analysis and flash (Gaganis & Varotsis, 2014).

Kashinath et al., 2018 extended the work of Gaganis & Varotsis, 2014, by proposing relevance vector machines (RVMs) and a single layer ANN for isothermal two-phase flash. In their work, the results were compared with a computationally expensive and challenging negative flash algorithm that distinguishes between the subcritical and supercritical regions using a cubic equation of state (EOS). In the proposed method, the first RVM is used to classify supercritical phases. A second RVM classifier is used to identify the number of stable phases in the sub-critical region. Finally, an ANN is used to predict the K-values for the given pressure and overall composition. The results of three different reservoir fluid mixtures with varying components showed that their proposed method could speed up phase-equilibrium calculations by 25 - 75% with less than 0.01% error compared to the negative flash method. The speed-up from using their algorithm could be increased to over 90% if the error margin increases to 4 - 5% (Kashinath et al., 2018).

Advancements in ANN algorithms and their handling of non-linear problems have made them the preferred ML model for researchers. K. Wang et al., 2019 used two ANN models to assist the traditional flash calculations in achieving faster and robust convergence. For a mixture with a known number of components, the mixture components' properties are used in ANNs to determine the upper and lower saturation pressures. The stability analysis is performed using the saturation pressures. If the system's pressure is between the maximum and minimum predicted pressure, the mixture is considered unstable; otherwise, it is considered stable in a single-phase state. The authors caution that the ANN model might give wrong stability results, but the wrong predictions have little effect on the simulation results. When the phase is unstable, and phase splitting is required, the second ANN model predicts initial guesses of the vapor mole fraction and K- values. The results from applying the two models echoed those of other researchers, with more than a 90% reduction of the time spent on stability analysis and flash calculations. However, contrary to other researchers' results, K. Wang et al., 2019 indicated that prediction accuracy is not affected by increasing the mixture's components in isothermal conditions.

The work discussed earlier focused on applying ML to speed up compositional fluid property modeling for conventional petroleum reservoirs. However, S. Wang et al., 2019 extended the application of ML to unconventional reservoirs with significant capillary effects by introducing the pore radius as an input into the ANNs. They developed a proxy flash model with two ANNs and implemented this in their in-house simulator. The proxy ANN model performed a two-step classification-regression process, where an ANN classifier is first used for phase classification, after which the phase mole fractions are determined using an ANN regression model. The input parameters for both neural networks are pressure, temperature, concentration, and pore radius. In their framework, the phase classification step results are accepted as final if the phase classification

step predicts that only one phase exists. However, if two phases exist in the system, the second ANN predicts the capillary pressure and K-values. These values are then used to initialize the physical flash calculation instead of Wilson's K-value correlation. Their results indicated that the implementation of the proxy ML model reduced the number of iterations for flash by more than 50%, increased the stability or convergence ratio by 8%, and reduced CPU time by 10 - 12% when compared with the initialization of the K-value using its value from the previous iteration. Their work also showed that the error increased as the number of components in the reservoir fluid increased.

Although the DL models are typically very fast compared to rigorous analytical or numerical modeling, one of their limitations lies in the fact that the model accuracy is stochastic, and there is no way to guarantee that output will honor physical laws. One way to address this limitation involves using physics-informed neural networks (PINNs) instead of the standard DL models. The application of PINNs in different engineering aspects has increased significantly over the years because they facilitate the seamless integration of data and physics into neural networks. They can be used to model systems that have governing equations based on the laws of physics. Much of the previously published use of PINNs include the solution of partial differential equations (PDEs) such as the Buckley Leverett equation (Fraces et al., 2020; Fuks & Tchelepi, 2020), Burgers' equation (Raissi, 2018; Raissi et al., 2019), Korteweg-De Vries (KdV) equation (Raissi, 2018, Raissi et al., 2019), Kuramoto-Sivashinsky equation (Raissi, 2018), non-linear Schrodinger (Raissi et al., 2019), and Navier Stokes equations (Raissi, 2018, Raissi et al., 2019), and the Allen–Cahn equation (Raissi et al., 2019).

Additionally, other studies have shown evidence of the feasibility of implementing PDE constraints as extensions to the computational graph for deep neural networks (Huang et al., 2020;

Xu & Darve, 2020). PINNs are well suited to the solution of PDEs because they can leverage the automatic differentiation used in the backpropagation step of the DL model training process to compute the differential operators in the PDEs (Raissi et al., 2019). This allows for the quick solution of complex multi-rate, multi-scale problems in solid mechanics (Haghighat et al., 2020) and fluid mechanics (Raissi et al., 2018, Raissi et al., 2019, Fraces et al., 2020). Although most publications on integrating physics into neural networks involve solving PDEs, the boundary and initial conditions are not PDEs. However, they meet the criteria of being differentiable and continuous (Daw et al., 2020). Since the main stipulation for using PINNs is that the physics-based equations added to the standard loss function are differentiable and continuous (Daw et al., 2020), PINNs can be developed with any physics constraints where the output is supposed to honor some physical laws (Manepalli et al., 2019; Yucesan & Viana, 2019; Daw et al., 2020; Dourado & Viana, 2020; Kashinath et al., 2021).

Data is an integral part of any machine learning involved process. Therefore, when machine learning algorithms are used to generate models for compositional modeling, experimental design for mixtures (i.e., mixture design) is used to ensure that the data used for the training covers the sample space. In experimental design, space-filling designs are used to place the design points to cover the entire experimental region (Joseph, 2016). The design space comprises three component mixtures with a sum of one, so each component is expressed as a fraction of the mixture. Statistical software like JMP (SAS Institute Inc, 2020-2021) can be used to implement the experimental design of the mixtures. JMP uses a constrained Fast Flexible Filling space-filling mixture design (Lekivetz & Jones, 2015) that evenly spreads the points over the design region, allowing us to sample the entire design region and set linear constraints (SAS Institute Inc, 2020-2021). The Fast Flexible Filling mixture design is a maximum projection design that maximizes space-filling

properties on projections to all subsets of factors (Joseph et al., 2015). This solves the issue of poor projections on lower-dimensional spaces resulting from minimax and maximin designs considering only space-filling in the entire dimensional space (Joseph et al., 2015).

3. VAPOR-LIQUID EQUILIBRIUM FOR MULTICOMPONENT SYSTEMS.

This chapter provides a background on the stability analysis and two-phase flash calculation processes that make up compositional modeling. This chapter covers how the Peng-Robinson equation of state (PR-EOS) can be used to describe the phase behavior of multicomponent systems using algorithms. The Phase behavior calculations involve determining the amounts of the equilibrium phases and their respective compositions by satisfying the equality of chemical potentials, inter-phase mass balance constraint, and the component balance constraint.

A mixture's volumetric and phase behavior can be described by cubic equations of state (EOS's) using the critical properties (critical temperature T_c and critical pressure P_c), acentric factor (ω), and interaction factor in the form of equilibrium ratios (K-values) of each component (Firoozabadi, 2016). The majority of the EOS applications rely on isothermal two-phase flash calculations, and the most common EOS used in petroleum applications is the Peng- Robinson Equation of state (PR-EOS).

For *N* components, $i = 1....N$, Figure 3.1 shows a two-phase system with a mixture comprising a feed composition (z_i) at pressure (P) and temperature (T). The amounts of each phase are denoted by the vapor fraction V and liquid fraction L. The composition of the liquid phase and vapor phase is denoted by x_i and y_i respectively.

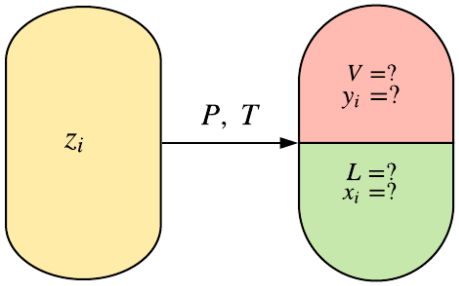

Figure 3.1. Two-phase system

Before determining the amounts and composition of the equilibrium phases, we first determine if the mixture is stable or unstable at the given temperature and pressure. If the mixture is stable in the single-phase state, it will exist in either the single-phase liquid or gas states. Otherwise, we assume it is stable in the two-phase state. The process of determining the mixture's stability is referred to as stability analysis (Whitson & Brulé, 2000). The overarching goal is to ensure that the mixture exists in the state to minimize Gibb's free energy. This is consistent with the second law of thermodynamics, which requires Gibb's free energy to be minimum at equilibrium.

Phase stability analysis determines if a mixture can attain a lower Gibbs free energy level by splitting the mixture into two or more phases and evaluating Gibb's free energy. Different algorithms have been designed to perform stability analysis. In this work, I use an algorithm based on the Michelsen stability test (Michelsen, 1982a, 1982b), which uses the tangent plane distance (TPD) analysis. The Michelsen stability test is based on locating second phase compositions with tangent planes parallel to the mixture composition (Whitson & Brulé, 2000). It involves two successive tests conducted separately, with one converging, assuming the second phase is liquid-like and the other converging, assuming the second phase is vapor-like (Whitson & Brulé, 2000). A summary of the Michelsen phase stability analysis procedure is shown in Figure 3.2.

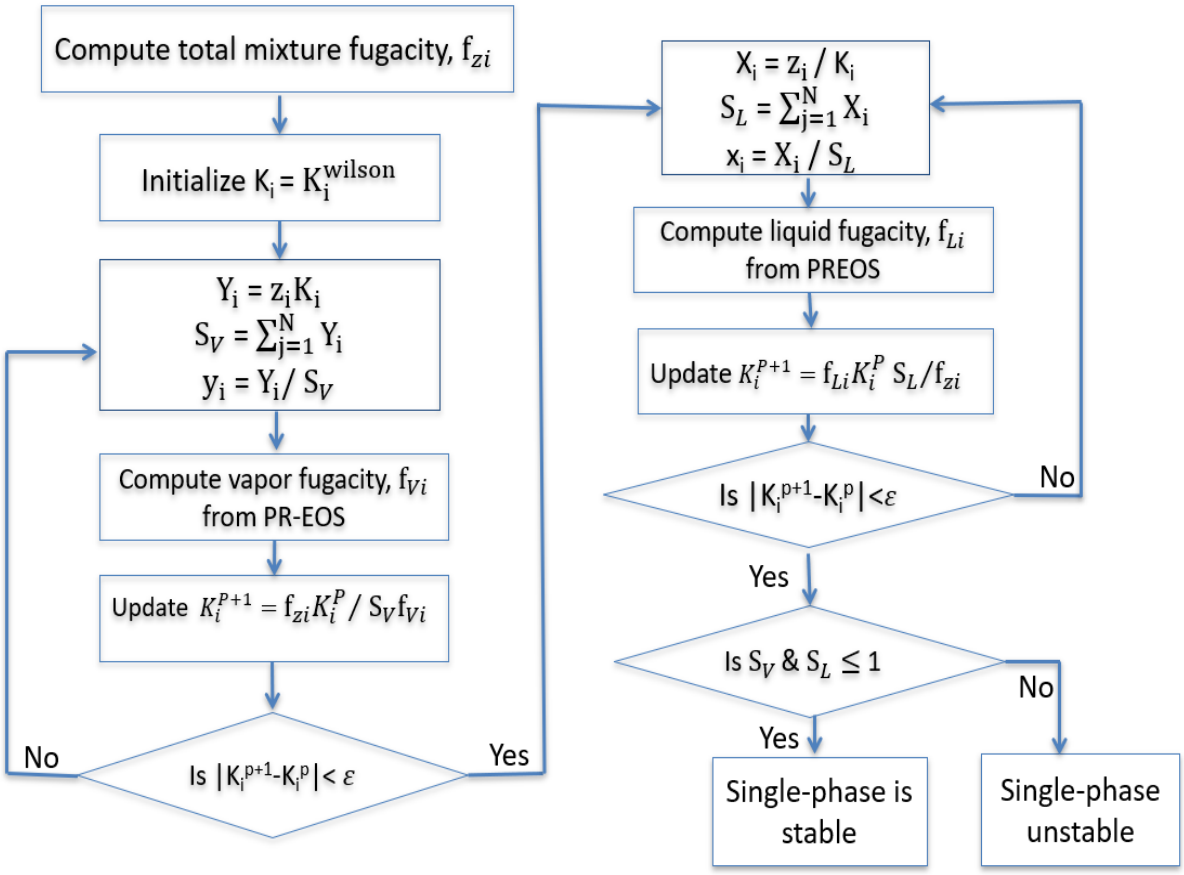

Figure 3.2. Flowchart for the phase stability analysis algorithm.

After determining the stability of the mixture, no further computation is required if it is stable in a single-phase state. However, if the mixture exists in a two-phase state, two-phase flash calculations are performed to determine each component's mole fractions in each phase. Figure 3.3 summarizes the procedure for the two-phase flash calculation used in this work.

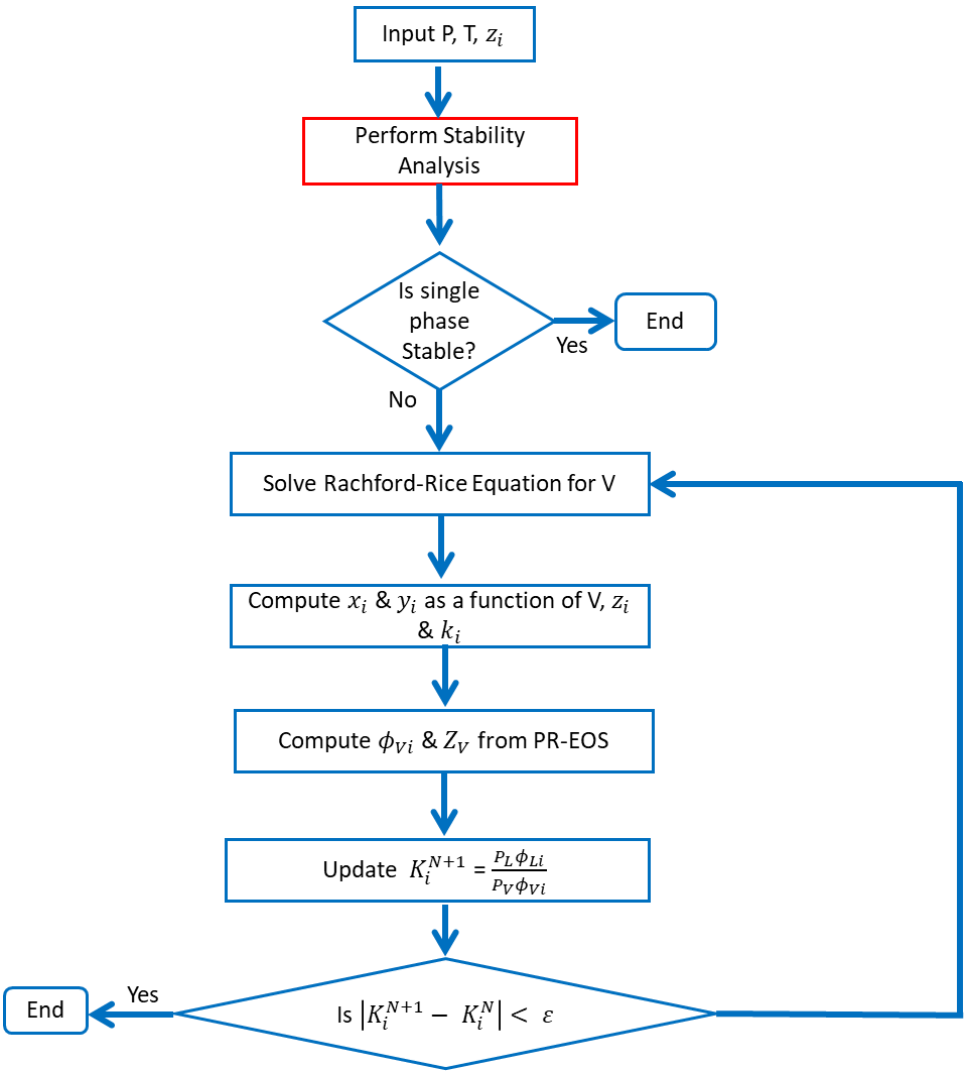

Figure 3.3. Flow chart for two-phase equilibrium alculations algorithm

Mathematically, two-phase flash can be solved using a successive-substitution or Newton-Raphson algorithm to ensure that

(i) the chemical potential of each component is the same in both phases,

(ii) the mole-fractions of each component in each phase, as well as the overall mole fraction of each component, sum up to one, and

(iii) the mass or number of moles of each component is conserved across the two phases. (Whitson & Brulé, 2000).

Each of these conditions is implemented as a constraint and discussed in the following subsection.

3.1. Chemical-potential (equal fugacity) constraint

This sub-section focuses on the phase equilibrium between the liquid (or oil) and vapor (or gas) phases, which can be formulated in terms of the fugacity of each hydrocarbon component in the liquid and vapor phases. The criterion of thermodynamic equilibrium is satisfied when the chemical potential of all the components in the liquid and vapor phases is equal, assuming all other forces are negligible (Whitson & Brulé, 2000). Chemical-potential is typically expressed in terms of fugacity.

Fugacity has all the features of chemical potential, but unlike chemical potential, it has an absolute value (Firoozabadi, 2016). So, it is typically used as a proxy for chemical potential in thermodynamics. The relationship between chemical potential μ_i and fugacity f_i can be written as (Firoozabadi, 2016):

$$d\mu_i = RT\ dlnf_i \quad [1]$$

For components, *i = 1....*, *N*, the equal fugacity constraint is given as:

$$f_{Li}(x_i, z_L, P_L, T) = f_{Vi}(y_i, z_V, P_V, T) \quad [2]$$

Where P_L and P_V are the pressures in the liquid and vapor phases, respectively. In this work, the fugacity of each component in the liquid (f_{Li}) and vapor phases (f_{Vi}) are calculated from the fugacity coefficients $\emptyset_i$ using the Peng-Robinson equation of state (Peng & Robinson, 1976). The phase compressibility factors Z_L and Z_V are also calculated with this equation of state (EOS). The Peng Robinson EOS (PR-EOS) is given as:

$$P = \frac{RT}{v-b} - \frac{a}{v(v+b)+b(v-b)} \quad [3]$$

Where *a* and *b* are constants given as:

$$a = 0.45724 \frac{\propto R^2 T_c^2}{P_c} \quad [4]$$

$$b = 0.07780 \frac{R^2 T_c^2}{P_c} \quad [5]$$

The symbols $\propto$, m, and ω represent a dimensionless function of reduced temperature and acentric factor, a constant characteristic of each substance, and the acentric factor, respectively. They are given as:

$$\propto = \left[1 + m(1 - \sqrt{\frac{T_c}{T}}\right]^2 \quad [6]$$

$$m = 0.37464 + 1.54226\omega - 0.26992\omega^2 \quad [7]$$

$$\omega = \frac{3}{7}\left(\frac{T_{bi}/T_{ci}}{1 - \frac{T_{bi}}{T_{ci}}} log\left(\frac{P_{ci}}{P_{sc}}\right) - 1\right) \quad [8]$$

Here, R is the gas constant, and T_c and P_c are the critical temperature and pressure, respectively. The PR-EOS is expressed in terms of the compressibility (Z) factor for the liquid and vapor phases as follows:

$$Z_L^3 - (1 - B_L)Z_L^2 + Z_L(A_L - 3B_L^2 - 2B_L) - (A_L B_L - B_L^2 - B_L^3) = 0 \quad [9]$$

$$Z_V^3 - (1 - B_V)Z_V^2 + Z_V(A_V - 3B_V^2 - 2B_V) - (A_V B_V - B_V^2 - B_V^3) = 0 \quad [10]$$

where:

$$A_L = \frac{aP_L}{(RT)^2} \quad B_L = \frac{bP_L}{RT} \quad Z_L = \frac{P_L V_L}{RT} \quad A_V = \frac{aP_V}{(RT)^2} \quad B_V = \frac{bP_V}{RT} \quad Z_V = \frac{P_V V_V}{RT} \qquad [11]$$

From the PR-EOS, the fugacity coefficients $ln\emptyset_{Vi}$ and $ln\emptyset_{Li}$ are given as:

$$ln\emptyset_{Vi} = ln\frac{f_{Vi}}{y_i P} \qquad [12]$$

$$ln\emptyset_{Li} = ln\frac{f_{Li}}{x_i P} \qquad [13]$$

where:

$$ln\emptyset_{Li} = \frac{b_{iL}}{b_L}(Z_L - 1) - ln\ ln\ (Z_L - B_L) + \frac{A_L}{2\sqrt{2B_L}}\left(\frac{b_{iL}}{b_L} - \frac{2}{a_L}\sum_{J=1}^{N} x_{jL}a_{ij})\right) ln\left[\frac{z_L + 2.414B_L}{Z_L - 0.414B_L}\right] \qquad [14]$$

$$ln\emptyset_{Vi} = \frac{b_{i,V}}{b_V}(Z_V - 1) - ln\ ln\ (Z_V - B_V) + \frac{A_V}{2\sqrt{2B_V}}\left(\frac{B_{iV}}{B_V} - \frac{2}{a_V}\sum_{J=1}^{N} y_{jL}a_{ij})\right) ln\left[\frac{Z_V + 2.414B_V}{Z_V - 0.414B_V}\right] \qquad [15]$$

$$a_{ij} = \left(1 - k_{ij}\right)\sqrt{a_i a_j} \qquad [16]$$

$$a_V = \sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j a_{ij} \qquad [17]$$

$$a_L = \sum_{i=1}^{N}\sum_{j=1}^{N} x_i x_j a_{ij} \qquad [18]$$

$$b_L = \sum_{i=1}^{N} x_i b_i \qquad [19]$$

$$b_V = \sum_{i=1}^{N} y_i b_i \quad [20]$$

Here, k_{ij} are the binary-interaction parameters between components *i* and *j*, whereas $\emptyset_{Vi}$ and $\emptyset_{Li}$ are the fugacity coefficients of each component in the vapor and liquid phases respectively.

3.2. Inter-phase Mass Balance Constraint

In phase equilibrium of multicomponent mixtures, it is essential that the mass or number of moles of each hydrocarbon component is conserved regardless of the state(s) in which the mixture is stable. The inter-phase mass balance constraint states that the total number of moles *n* of the feed composition z_i should distribute into n_L moles of liquid with composition x_i and n_v moles of vapor with composition y_i without loss of matter or chemical alteration of the component species (Whitson & Brulé, 2000).

The overall mole fraction of each component (z_i) can be written as a function of the phase mole fractions (x_i and y_i) and vapor fraction (V) as follows:

$$z_i = V y_i + (1 - V) x_i \quad [21]$$

The phase mole fractions x_i and y_i are obtained by combining equation [21] with the definition of the vapor-liquid equilibrium factor ($k_i = y_i/x_i$) as follows:

$$x_i = \frac{z_i}{V(k_i - 1) + 1} \quad [22]$$

$$y_i = \frac{z_i k_i}{V(k_i - 1) + 1} = k_i x_i \quad [23]$$

The K-values (k_i) for the iterative solution of the Rachford-Rice (RR) equation are obtained from Wilson's correlation (Wilson, 1968):

$$k_i^0 = \frac{exp\,[5.37(1+\omega_i)(1-(T_{ci}/T)]}{P/P_{ci}} \quad [24]$$

3.3. Component Balance Constraint

The final constraint to be satisfied in two-phase flash calculations is the component balance constraint. It requires that every component's overall mole fractions and phase mole fractions must sum up to one. This is because each component's total number of moles in each phase and in the overall mixture is conserved. Recalling that:

$$x_i = \frac{n_L^i}{n_L}, \qquad y_i = \frac{n_G^i}{n_G}, \quad and\ z_i = \frac{n_i}{n} \quad [25]$$

where n_L^i is the number of moles of component *i* in the liquid phase, n_G^i is the number of moles of component *i* in the gas/vapor phase, and n_i is the number of moles of component *i* in the overall mixture. It is easy to show that each of these mole fractions will sum up to one:

$$\sum_{i=1}^{n_c} x_i = 1, \quad \sum_{i=1}^{n_c} y_i = 1, \quad \sum_{i=1}^{n_c} z_i = 1 \quad [26]$$

where n_c is the total number of components in the mixture.

Equation [26] is the mathematical representation of the component balance constraint. For computational efficiency, the first two can be combined and written as (MøYNER, 2021):

$$\sum_{i=1}^{n_c} (x_i - y_i) = 0 \quad [27]$$

Combining equations [22] and [23], we obtain the Rachford-Rice (RR) equation:

$$RR = \sum_{i=1}^{N}(x_i - y_i) = \sum_{i=1}^{N}\frac{z_i(k_i - 1)}{V(k_i - 1) + 1} = 0. \quad [28]$$

The Rachford-Rice (RR) equation solution yields the vapor mole fraction, V.

In a two-phase flash algorithm like the one shown in Figure 3.3, the algorithm begins with a Michelsen stability test (Michelsen, 1982a, 1982b) to assess whether the fluid mixture is stable at the given pressure and temperature in the single phase. If the fluid is stable in a single phase, the phase mole fractions are determined without solving the Rachford- Rice equation. Otherwise, a combination of successive substitution and Newton- Raphson's methods are used to compute the vapor fraction from equation [28] -the Rachford-Rice equation.

The phase mole fractions (x_i and y_i) are then estimated using equations [22] and [23] based on an initial estimate of the vapor-liquid equilibrium factor (K-value) derived using equation [24] - Wilson's correlation (Wilson, 1968). Next, the fugacity coefficient of the vapor phase $\emptyset_{Vi}$ and the compressibility of the vapor phase Z_{Vi} are computed from the PRE-EOS using equation [10] and equation [12] respectively. The K-values initialized with Wilson's correlation in equation [24] can be updated as follows:

$$k_i^{N+1} = \frac{P_L \emptyset_{Li}}{P_V \emptyset_{Vi}} \quad [29]$$

The algorithm refines the solution iteratively until convergence. The condition for convergence ε of the equilibrium ratio is specified as:

$$\left|K_i^{N+1} - K_i^N\right| \leq \varepsilon \quad [30]$$

(N) and (N +1) indicate the iteration level and the recommended convergence tolerance is $\varepsilon = 1.0e^{-13}$ (Whitson & Brulé, 2000) . If convergence is not reached, the K values are updated with successive substitution (Whitson & Brulé, 2000).

4. DEEP LEARNING AND INCORPORATION OF PHYSICS

Advancements in computational resources and deep learning, also known as artificial neural networks, have revolutionized problem solving in industries like health care, finance, search engines, etc. In recent years, there has been a surge in interest in using deep learning to solve engineering problems. The increased interest has led to the development of platforms like Theano (Bergstra et al., 2010), Tensorflow (Abadi et al., 2016), and MXNET (Chen et al., 2015) that facilitate deep learning calculations using techniques such as automatic differentiation (Baydin et al., 2018). There has also been an introduction of a new type of neural network called the Physics Informed Neural Network. These networks are specifically designed for engineering problems because they can be trained with data while integrating the equations governing the physics of the modeled systems. In this chapter, we review deep learning models and how standard DNNs can be modified to incorporate physical constraints using physics-informed neural networks (PINNs).

In this work, we used Tensorflow 2 to create the deep neural network models. TensorFlow provides the numeric platform that implements the mathematical operations needed for deep learning. This work focuses on predictive modeling using supervised machine learning. In this process, a model is created when a deep learning algorithm learns the target function that best maps the input variables to a training dataset's output variables to make predictions on any given new input variables (Brownlee, 2019). A better estimate of the target function provides better predictions from the model, so much time is spent attempting to improve this function through optimization. Since most of the recent work involving machine learning for compositional modeling is done using neural networks, this work will extend the work discussed in the literature review.

4.1. Deep Neural Networks (DNN)

This sub-section will briefly review deep neural networks and how a neural network generates one or more output value(s) from multiple input values. A more in-depth discussion on DNNs can also be found in renown textbooks in this field by Goodfellow et al., 2016, Trask, 2019 and Weidman, 2019.

A neural network is a non-parametric machine learning algorithm that does not make assumptions about the mapping function's form to learn and make predictions from non-linear training data. The neural network is essentially a universal approximator because of its ability to mathematically capture and map very complex relationships (Brownlee, 2019). Structurally, a neural network is a collection of neurons arranged in layers. Figure 4.1 shows a simple neural network structure made up of a group of neurons arranged in layers. The neurons in this layer do not perform any computation; they are simply a gateway for the input value to move to the next layer. The hidden layer(s) is/are the layer(s) between the input and output layers. The neurons in the hidden layer take as input the values from the input neurons and other preceding hidden layers. The output layer is the last layer, which obtains input from the last hidden layer and yields the output variable(s). The tremendous increase in computational power over the last couple of decades has enabled the application of deep neural networks with several layers.

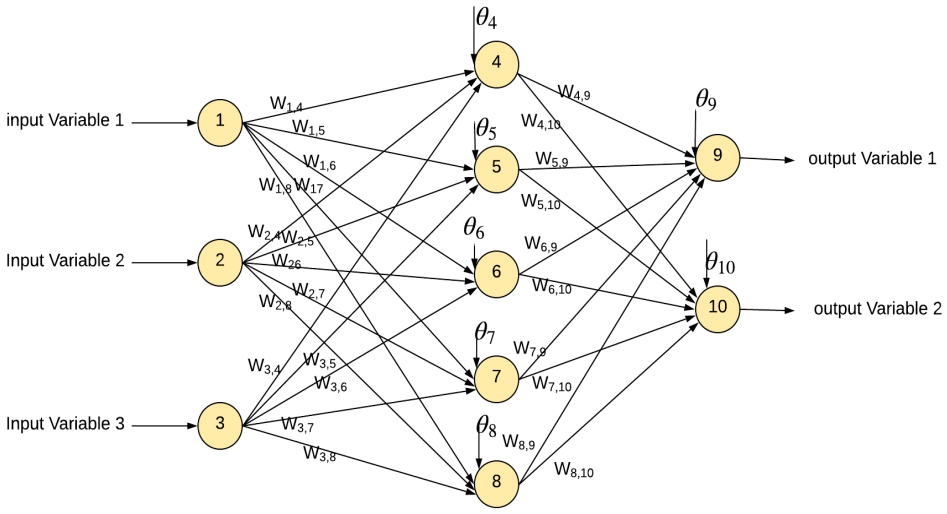

Figure 4.1. Configuration of a simple neural network

Figure 4.2 shows the configuration of a neuron. Each neuron has an input x_i, output y_j, weight w_{ij}, activation function a_i and bias. The weight ($w_{i,j}$) determines the strength and sign of the connection between a pair of neurons. The bias is represented as a dummy input ($a_0 = 1$) with a corresponding weight ($w_{0,j}$).

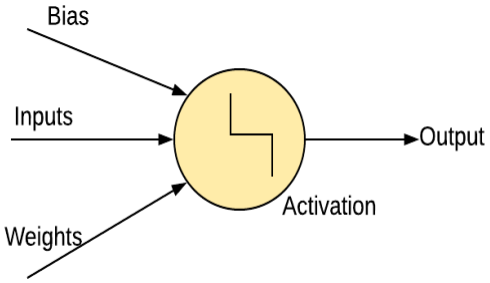

Figure 4.2. Configuration of a neuron

The output of a neuron in a single layer feed-forward neural network is computed as the weighted sum of the inputs as follows:

$$s_j = \sum_{i=1}^{n} w_{ij} a_i, \quad [31]$$

Applying the activation function σ to the weighted sum gives the output a_j of the neuron as:

$$a_j = \sigma\left(\sum_{i=1}^{n} w_{ij} a_i\right), \quad [32]$$

where:

$$a_i = \left(w_{0,j} x_i + a_0\right) + a_{i,j} \quad [33]$$

The activation function maps the weighted sum over a specified interval before passing it on to another neuron in the next layer. The primary purpose of the activation function is to help the neural network learn non-linear trends by determining the threshold of the neuron and the output signal's strength.

The two widely used activation functions are sigmoid and hyperbolic tangent (TanH). As shown in Figure 4.3, the TanH maps values to the [-1,1] interval and is mathematically represented by:

$$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad [34]$$

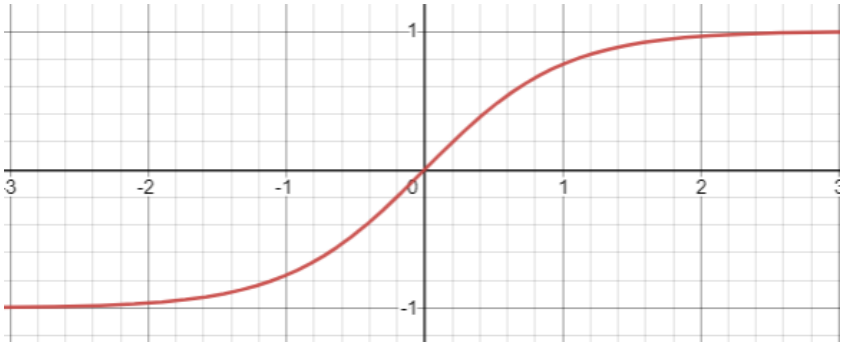

Figure 4.3. Hyperbolic tangent (Tanh) activation function

Figure 4.4 shows the sigmoid activation function. The sigmoid function maps values to the [0,1] interval. It is mathematically represented by:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad [35]$$

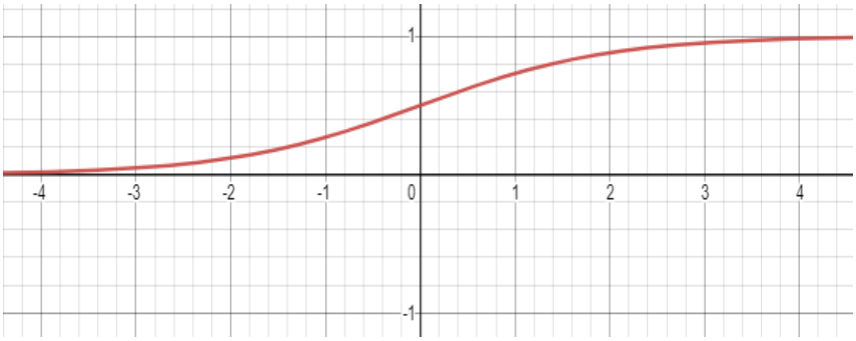

Figure 4.4. Sigmoid activation function

One of the most significant drawbacks of the sigmoid and hyperbolic functions is that the magnitude of the error diminishes drastically as the number of layers increases. This situation is often referred to as the vanishing gradient problem. The Rectified Linear Units (ReLU) activation function was introduced to overcome the vanishing gradient problem and has since become the most popular function for hidden layers. This is because it has a more straightforward

mathematical operation, which trains several times faster than the other activation functions while avoiding the vanishing gradient problem.

Figure 4.5 shows the ReLU activation function that maps values to the interval [0, x] and is represented by:

$$\sigma(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \quad [36]$$

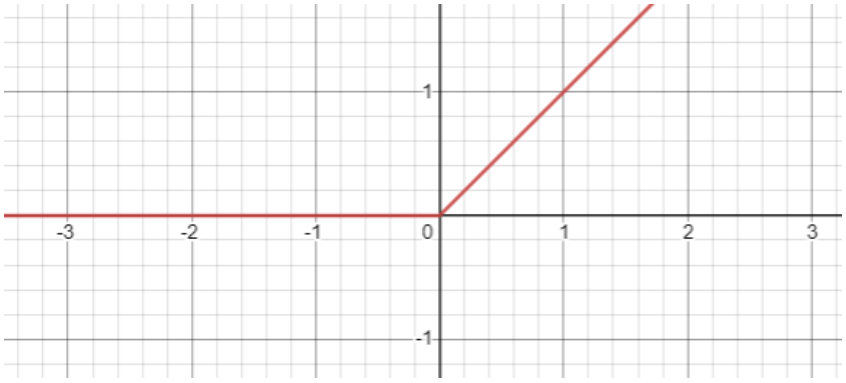

Figure 4.5. Rectified Linear Units (ReLU) activation function

It is worth noting that, unlike the other activation functions, the ReLU activation function is only applicable to hidden layers.

The type of activation function used for the output layers depends on the type of problem being solved. For classification problems, the SoftMax activation function is used. This activation function maps the value to the interval [0,1]. It is analogous to a categorical probability distribution in that it splits each result so that the sum of all outputs equals 1. For the regression problems, any other activation functions can be used for the output layer. The two most popular are hyperbolic tangent function and sigmoid function.

Training the neural network.

Training a model involves estimating the weights that lead to the best predictive results by minimizing the loss on the dataset using gradient descent. A standard method for estimating weights in neural networks involves iteratively updating weights based on the error of the output node.

The most widely used approach of utilizing model errors to update weights is referred to as Back Propagation of Error. In this approach, the error of the output node $\triangle_p$ is distributed across all hidden nodes *j* that led to it. The determined error values $\triangle_j$ for the hidden layer, the error is based on the connection's strength between the hidden node and the output node.

For a neural network with *p* nodes in the output layer, the error is defined as (Russell & Norvig, 2016):

$$\triangle_p = \left(y_p - \widehat{y_p} \right) \sigma'(s_p).$$

$$\triangle_p = \left(y_p - \widehat{y_p} \right) \sigma' \left(\sum_{i=1}^{n} w_{i,p} a_i \right). \quad [37]$$

The weight update rule for minimizing the loss at the output layer is specified as (Russell & Norvig, 2016):

$$w_{i,j}^{new} = w_{i,j}^{old} + l(a_j \triangle_p) \quad [38]$$

l is the learning rate, weight decay parameter, or step size that determines how the weights change from one iteration to another. The learning rate *l* can be configured to remain constant over time or to degrade as the learning process progresses.

During back propagation of the error, $\triangle_p$, the error of the hidden layers $\triangle_j$ is distributed based on the strength of the connection between the hidden node and the output node.

The error values of the hidden layers are computed as (Russell & Norvig, 2016):

$$\triangle_j = \sigma'(s_j) \sum_k w_{j,k} \triangle_p$$

$$\triangle_j = \sigma' \left(\sum_{i=1}^{n} w_{ij} a_i \right) \sum_p w_{j,p} \triangle_p \quad [39]$$

The $\triangle_j$ error values at each node are used to update the weights using the weight update rule for minimizing the loss in the hidden layers, which is defined as (Russell & Norvig, 2016):

$$w_{i,j}^{new} = w_{i,j}^{old} + l(a_i \triangle_j) \quad [40]$$

This rule is similar to equation [38], with *l* as the learning rate.

The loss function utilized in this work is the mean-squared error (MSE) which is defined as:

$$L = \frac{1}{N} \sum_{j=1}^{N} (y_j - \hat{y}_j)^2 \quad [41]$$

where y_j is the actual output value and $\hat{y}_j$ is the predicted output value.

For a neural network with *p* nodes in the output layer, the loss at the *p*th output unit is:

$$L_p = (y_p - a_p)^2 \quad [42]$$

To minimize this function using a gradient-based (or stochastic gradient) approach, we need to find the gradient of the loss $L_p$with respect to the weights $w_{j,p}$ connecting the hidden layer to the *p*th output unit. To determine the gradient, we differentiate the loss function with respect to the weight $w_{j,p}$ as follows (Russell & Norvig, 2016):

$$\frac{\partial L_p}{\partial w_{j,p}} = -2(y_p - a_p) \frac{\partial a_p}{\partial w_{j,p}},$$

$$= -2\left(y_p - a_p\right)\frac{\partial s_p}{\partial w_{j,p}},$$

$$= -2\left(y_p - a_p\right)\frac{\partial s_p}{\partial w_{j,p}},$$

$$= -2\left(y_p - a_p\right)\sigma'\left(s_p\right)\frac{\partial s_p}{\partial w_{j,p}},$$

$$= -2\left(y_p - a_p\right)\sigma'\left(s_p\right)\frac{\partial}{\partial w_{j,p}}\left(\sum_j w_{j,p}a_j\right),$$

$$= -2\left(y_p - a_p\right)\sigma'\left(s_p\right)a_j,$$

$$\frac{\partial L_p}{\partial w_{j,p}} = -a_j \triangle_p \qquad [43]$$

To determine the gradient of the loss with respect to the weights connecting the input layer to the hidden layer $w_{i,j}$, we differentiate the loss function with respect to the weight $w_{i,j}$ as follows(Russell & Norvig, 2016):

$$\frac{\partial L_p}{\partial w_{i,j}} = -2\left(y_p - a_p\right)\frac{\partial a_p}{\partial w_{i,j}},$$

$$= -2\left(y_p - a_p\right)\frac{\partial s_p}{\partial w_{i,j}},$$

$$= -2\left(y_p - a_p\right)\sigma'\left(s_p\right)\frac{\partial s_p}{\partial w_{i,j}},$$

$$= -2\triangle_p \frac{\partial}{\partial w_{i,j}}\left(\sum_j w_{j,p}a_j\right),$$

$$= -2\triangle_p w_{j,p}\frac{\partial a_j}{\partial w_{i,j}},$$

$$= -2 \triangle_p w_{j,p} \frac{\partial \sigma(s_j)}{\partial w_{i,j}},$$

$$= -2 \triangle_p w_{j,p} \sigma'(s_j) \frac{\partial s_j}{\partial w_{i,j}},$$

$$= -2 \triangle_p w_{j,p} \sigma'(s_j) \frac{\partial}{\partial w_{i,j}} \left(\sum_{i=1}^{n} w_{ij} a_i \right),$$

$$= -2 \triangle_p w_{j,p} \sigma'(s_j) a_i,$$

$$\frac{\partial L_p}{\partial w_{i,j}} = a_i \triangle_j \quad [44]$$

The process of updating the weights to minimize the loss is an optimization problem solved using gradient descent. During gradient descent, small values are used as the starting weights and then reduced until convergence on the minimum value of the loss. The gradient descent process can be either a batch process or a stochastic process. In batch gradient descent, we minimize the sum of the entire dataset for each learning rate. This process guarantees convergence to a global minimum if the learning rate is low enough; however, it tends to be very computationally expensive. To speed the training process up, stochastic gradient descent can be used. In stochastic gradient, we minimize the sum of a subset of the dataset at each learning rate, thereby accelerating the training process. However, one shortcoming of stochastic gradient training is that it does not guarantee convergence, but this can be overcome by steadily decreasing the learning rates as the training process proceeds.

4.2. Incorporating Physics into Neural Networks

This sub-section will discuss how physical laws/constraints are incorporated into standard neural networks to obtain the so-called "physics-informed neural networks" (PINNs). Unlike most

previous publications that include physics-based constraints as a simple arithmetic sum of additional loss functions to the standard loss function, this work will use a weighted-summation in this penalty-based approach. To determine the optimum weights that simultaneously maximizes the accuracy of the prediction and enforce the physical constraints, several cases were run at different weight values, as discussed in Chapter 6.

As discussed in section 0, when using deep learning algorithms like neural networks to model systems that have governing equations based on the laws of physics, the model can yield predictions that do not honor physics unless the physical laws are explicitly integrated into the learning process. Numerous authors have created several methods for integrating physics into deep neural networks. In this work, I use penalty-based PINNs, which are typically used to incorporate PDE constraints into the training of deep learning models. According to Raissi et al., 2019, this technique adds the mean-squared errors (MSE) associated with the governing PDEs, as well as the initial and boundary conditions, to the standard loss function defined in equation [41]. The modified loss function is presented as:

$$L = MSE_1 + MSE_2 + MSE_3, \qquad [45]$$

where MSE_1 , MSE_2 , and MSE_3 are the MSEs associated with the standard DL, PDE, and boundary conditions, respectively. It is worth noting that Raissi et al., 2019 proposed a different version of the equation that included the loss related with the PDE. (MSE_2) only instead of both MSE_2 and MSE_3 . In this multi-objective optimization problem, the strategy of simply adding up the MSEs implicitly assumes equal weights for all three MSEs. Given that the magnitudes of each MSE may differ, there is no guarantee that all three MSEs will be minimized to the same extent. The greatest of these MSEs will typically be minimized at the expense of the others. This work proposes using

a weighted summation of the standard DL MSE and the MSEs associated with the thermodynamic constraints. Due to the computational complexity of determining fugacity equality and the requirement to solve the equation of state for each fluid combination, we focus on the interphase mass balance and component balance constraints.

To be clear, generating training data from the iterative solution of the RR equation is analogous to generating training data for traditional PDE-based PINNs using a numerical or analytical model, as described in Raissi et al., 2019 and Haghighat & Juanes, 2021 (Ihunde & Olorode, 2022). Therefore, we use the interphase mass balance and component balance constraints in place of the PDEs and boundary conditions in Raissi et al., 2019 and Haghighat & Juanes, 2021 (Ihunde & Olorode, 2022). Given that the mathematical formulation and usage of the penalty-based approach is not limited to PDEs, this work will demonstrate the feasibility of introducing physical constraints that are not PDEs into a neural network (Ihunde & Olorode, 2022).

The modified loss function used in this work is given as:

$$Loss = \lambda_1 MSE_1 + \lambda_2 MSE_2 + \lambda_3 MSE_3 \quad [46]$$

where these three loss functions are given as:

$$MSE_1 = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(y_{i,j} - \widehat{y_{i,j}}\right)^2, \quad [47]$$

$$MSE_2 = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n_c}\left(x_{ij}(1 - V_i) + y_{ij}V_i - z_{ij}\right)^2, \quad [48]$$

$$MSE_3 = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n_c}\left(x_{ij} - y_{ij}\right)^2, \quad [49]$$

and λ_1, λ_2, and λ_3 are their corresponding weights. Note that the inner summation in equation [47] is over the total number of variables M, whereas it is over the total number of components n_c in equations [48] and [49] (Ihunde & Olorode, 2022). A comparison of the equations for MSE_2 and MSE_3 with the equations for the interphase mass balance and component balance constraints (equations [21] and [26]) indicates that these equations correspond to both constraints in the homogeneous form (Ihunde & Olorode, 2022). On the other hand, MSE_1 is the data misfit, which is based on the difference between the model and the training data, as in equation [41]. To determine the importance of MSE_2 and MSE_3, I run them separately in this work to avoid one disguising the influence of the other. This means there are two basic scenarios: one with a loss function that just includes MSE_1 and MSE_2, and the other with MSE_1 and MSE_3.

4.3. Model evaluation metrics

Since the standard metrics used to evaluate model performance strictly evaluate if the prediction made by the model is close to the actual value in the training data, we need a metric that will evaluate the extent to which the model predictions honor the physical constraints. In this sub-section, we discuss how to evaluate the effectiveness of the PINN model at honoring the physical constraints using the root mean squared error (RMSE) (Ihunde & Olorode, 2022).

For the interphase mass balance constraint, the RMSE is given as:

$$RMSE_2 = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n_c}\left\{\left(z_{i,j} - \left[\hat{x}_{i,j}\left(1-\hat{V}_i\right) + \hat{y}_{i,j}\hat{V}\right]\right)^2\right\}}{N}} \qquad |i = 1,2,3 \ldots\ldots N \tag{50}$$

Considering that the overall mole fractions provided in the training, validation, and test data are guaranteed to sum up to one, the component balance constraint will consist of two RMSEs—one

for the liquid phase ($RMSE_L$) and the other for the vapor phase ($RMSE_V$) (Ihunde & Olorode, 2022). These two RMSEs are computed as follows:

$$RMSE_L = \sqrt{\frac{\sum_{i=1}^{n}\left(\left(\sum_{j=1}^{n_c}\hat{x}_{i,j}\right)-1\right)^2}{N}} \qquad |i = 1,2,3 \ldots\ldots N \tag{51}$$

$$RMSE_V = \sqrt{\frac{\sum_{i=1}^{n}\left(\left(\sum_{j=1}^{n_c}\hat{y}_{i,j}\right)-1\right)^2}{N}} \qquad |i = 1,2,3 \ldots\ldots N \tag{52}$$

To keep things simple, we sum these two RMSEs to obtain a single RMSE for the component balance constraint as follows:

$$RMSE_3 = RMSE_L + RMSE_V \tag{53}$$

5. IMPLEMENTATION

This chapter discusses how the PINN and standard DNN models are designed and implemented. It begins with a discussion of how the data used to train the models is generated. We review the use of a space-filling mixture design to generate one million overall compositions (z_i) that are evenly distributed throughout the sample space (Ihunde & Olorode, 2022). Next, we discuss how MATLAB Reservoir Simulation Toolkit (MRST) (Lie, 2019) was used to perform stability analysis and isothermal two-phase flash to obtain the phase mole-fraction values x_i and y_i and the vapor fraction V (Ihunde & Olorode, 2022). The overall composition, pressure, phase mole-fractions, and vapor fraction make up the dataset used to train the PINN and DNN models (Ihunde & Olorode, 2022). This chapter ends with a discussion of how the standard DNN model and PINN models are implemented.

5.1. Experimental Design for Mixtures

Data is a critical component of every machine learning process. When machine learning methods are used to generate models for compositional modeling, experimental design for mixtures (i.e., mixture design) is used to ensure that the training data covers the sample space.

In this work, the JMP software is used for the experimental design for mixtures. Ten thousand unique compositions of three-component reservoir fluids are generated using a space-filling mixture design in JMP (Ihunde & Olorode, 2022). This allows us to sample the entire parameter space $(0 \leq z_i \leq 1)$ evenly while enforcing the linear constraint that requires that the mole-fractions of each component in each fluid mixture sums up to one $(\sum_i^3 z_i = 1)$. The fluid mixtures used to test the feasibility of incorporating physics-based components include methane (C1), propane (C3), and tetradecane (C14). **Error! Reference source not found.** shows a ternary plot

of some of the compositions obtained using the space-filling mixture design. It shows that the use of the design yields mixtures that are evenly distributed within the entire sample space of all possible mixtures with C1, C3, and C14.

In JMP, the fast flexible filling space filling design (Lekivetz & Jones, 2015) algorithm generates many random points within the specified design region (SAS Institute Inc, 2020-2021). Then, a Fast Ward algorithm (Ward Jr, 1963) is used to cluster the points into multiple clusters that equal the number of runs specified (SAS Institute Inc, 2020-2021). The final design points are obtained using the maximum projection (MaxPro) optimality criterion (SAS Institute Inc, 2020-2021). The MaxPro criterion maximizes the space-filling properties on projections to all subsets of factors, thereby generating good space-filling properties on factor projections. (Joseph et al., 2015).

For design points x_{ik} and x_{jK}, the MaxPro criterion ($C_{Max\ Pro}$) to construct an n-run design in p factors is defined as (Joseph et al., 2015):

$$C_{Max\ Pro} = \sum_{i}^{n-1} \sum_{j=i+1}^{n} \left[\frac{1}{\prod_{k=1}^{p} (x_{ik} - x_{jK})^2} \right] \quad [54]$$

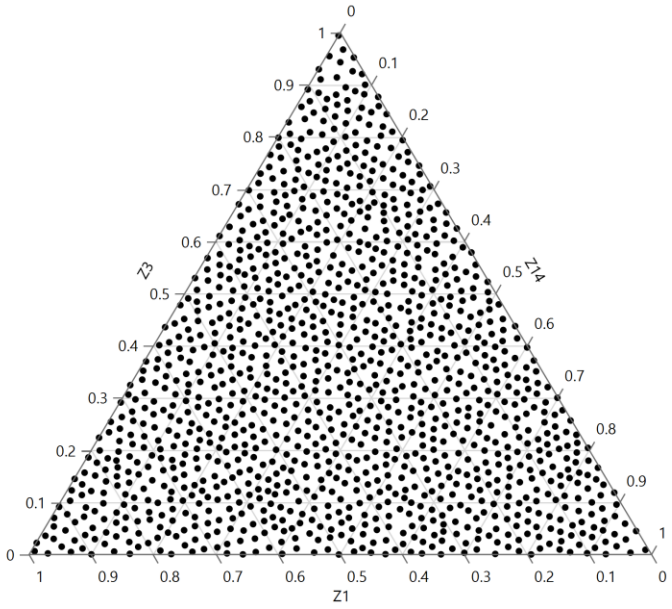

Figure 5.1. Ternary plot showing some of the compositions

5.2. Compositional modeling for data generation.

This subsection details the generation of the datasets used to train, validate, and test the DL models for two-phase flash. The compositional modeling routines in the MATLAB Reservoir Simulation Toolkit (MRST) (Lie, 2019) were used to perform stability analysis and isothermal two-phase flash (Ihunde & Olorode, 2022). The ten thousand compositions obtained from the space-filling design discussed in the previous section were flashed using 100 different pressures ranging between 14.7 and 5000 psia to generate one million fluid mixtures that make up the training dataset. Considering the focus on isothermal two-phase flash, the temperature was maintained at a value of 353 K (176°F). The two-phase flash procedure discussed in chapter 0 is used to obtain the phase mole fractions and vapor fraction for each fluid mixtures the dataset.

Table 1 shows the first six rows of a randomly selected subset of the data. The first four columns show the overall mole fractions (z_i) and pressure (P), which serve as input variables for the actual phase-equilibrium calculations (Ihunde & Olorode, 2022). The output variables from the two-

phase flash computations are shown in the remaining columns. The goal of this research is to use input data (overall mole fractions and pressure) to train deep neural networks (with and without physical restrictions) to predict output data (phase mole fractions and vapor fractions). The subscripts 1, 3, and 14 correspond to methane (C1), propane (C3), and tetradecane (C14), which are the hydrocarbon components in each of the fluid mixtures. The one million unique fluid mixtures were divided into training, validation, and testing in the ratio 70:15:15. During the training process, the training and validation datasets are used, while the test dataset is withheld and not exposed to the models.

Table 1. Sample from the dataset used to train DNN models.

z_1	z_3	z_{14}	P (psi)	V	x_1	x_3	x_{14}	y_1	y_3	y_{14}
0.0303	0.8517	0.1180	4345.358	0	0.0303	0.8517	0.1180	0	0	0
0.2450	0.3969	0.3580	1626.076	0	0.2450	0.3969	0.3580	0	0	0
0.9031	0.0937	0.0032	367.1495	1	0	0	0	0.9031	0.0937	0.0032
0.6841	0.3151	0.0009	1676.433	1	0	0	0	0.6841	0.3151	0.0009
0.6861	0.1365	0.1774	820.3632	0.7027	0.5919	0.1572	0.2509	0.9087	0.0876	0.0036
0.6125	0.2196	0.1679	1072.149	0.3554	0.2165	0.3113	0.4722	0.8308	0.1691	0.0001

Table 2 shows a description of the variables in the dataset. Three additional variables are added to the dataset to identify each composition as single-phase liquid, single-phase vapor, or two-phase. Figure 5.2 shows the code used to implement the label encoding and one-hot encoding used to generate the additional phase identifying variables. Using the vapor fraction V of each composition, label encoding is first performed to assign each composition with an integer value of "0", "1," or "2," identifying it as single-phase liquid, single-phase vapor, or two-phase. Since the integer values are only used as labels, one-hot encoding replaces the integer values with binary

variables to ensure that the model does not assume an ordinal relationship incorrectly. The possible classes for the fluid phase necessitate the use of three binary variables, each with a value of "1" for the identified phase and a value of "0" for the two other phases.

Table 2. Description of the variables in the dataset

z_1, z_3, z_{14}	Overall mole fraction for methane, Propane and Tetradecane
P	Pressure
V	Vapor fraction
x_1, x_3, x_{14}	Liquid-phase mole fraction for methane, Propane and Tetradecane
y_1, y_3, y_{14}	Vapor-phase mole fraction for methane, Propane and Tetradecane
liquid (1,0,0)	One-hot encoded variable identifying liquid phase
gas (0,1,0)	One-hot encoded variable identifying gas phase
two-phase (0,0,1)	One-hot encoded variable identifying two-phase

```
#label enconding to assign each phase with an integer value.
dataset['Phase'] = [2 if (l%1 !=0) else l for l in dataset['V']]

#one-hot enconding to replace the integer value with binary variables.
dataset = pd.concat([dataset,pd.get_dummies(dataset['Phase'],prefix='Phase')],axis=1)
dataset= dataset.drop(['Phase'],axis=1)
```

Figure 5.2. Snippet of code used to perform Label encoding and one-hot encoding.

Since the dataset contains variables of different magnitudes, the data is scaled to a range of [0, 1] using the "MinMaxScaler" from the Sci-kit learn Python package. The formula for "MinMaxScaler" is given as:

$$X_{std} = \frac{(X - \min)}{(\max - \min)}$$

This work uses the MinMaxScaler because the data has no outliers, and its standard deviation is relatively small.

5.3. Implementation of the Neural Network Models

This subsection discusses how the neural network models are implemented in this work. It starts with presenting the structure and parameters of the classification neural network model trained for phase identification. Next, we review how the standard DNN and PINN models are implemented. Lastly, considering the stochastic nature of the deep learning training process, section 5.4 describes how K-fold cross-validation is performed to guarantee the model's mean and standard deviation are reliably estimated.

The methodology used to implement the PINN and DNN models mimics the compositional fluid modeling process of performing a stability analysis for phase identification, then two-phase flash to determine phase compositions and vapor fraction. This process translates into a two-step process in ML with classification and regression, as shown in Figure 5.3. A classification model is developed in the first step to determine the phase from the molar compositions and pressure. DNN and PINN regression models are created in the second step using the input and output variables from the classification model as shown in Figure 5.3. This work aims to incorporate physics into the second step, as demonstrated using the PINN model. Figure 5.3 shows the neural network schematics with the input and output layers indicating the variables of interest.

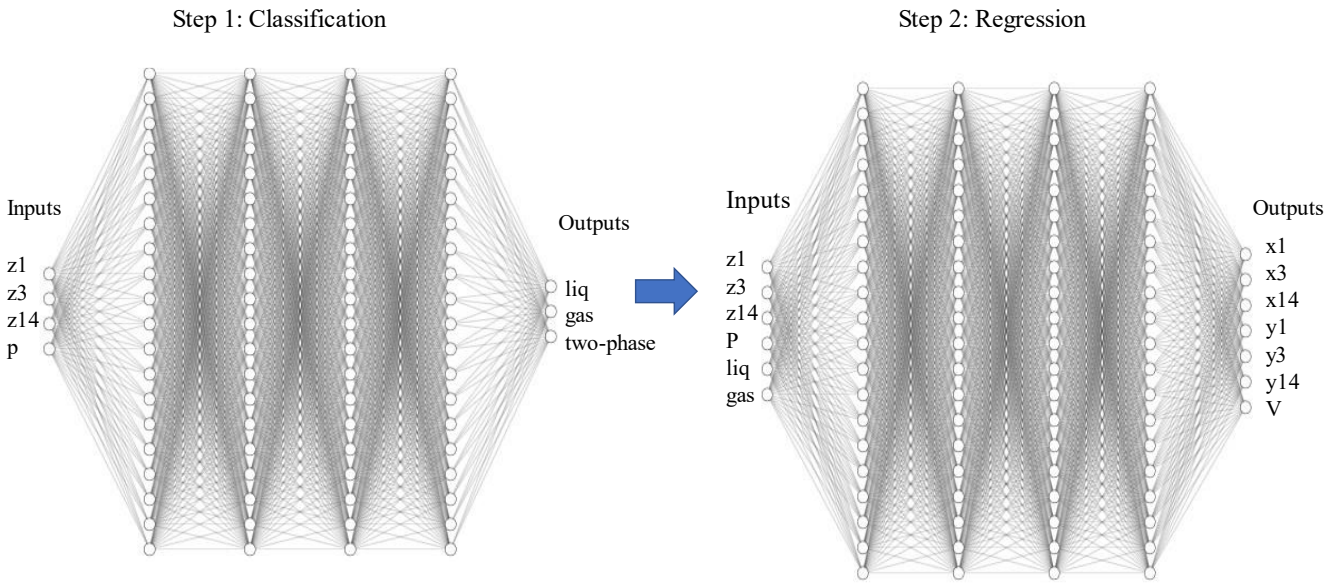

Figure 5.3. Neural network schematics showing the input and output variables

5.3.1 Implementation of the classification model.

Table 3 summarizes the key parameters of the classification neural network used to identify the state of each fluid mixture in the dataset. The network has four layers. The inputs of the network are the overall mole fractions (z_i) and the pressure *(p)*. The network is compiled with the Adam optimizer and the categorical cross-entropy loss function. The activation function for all the hidden layers is ReLU, and for the output layer, the SoftMax activation function. The network's output is a binary output that identifies the composition as liquid, gas, or two-phase. The SoftMax activation will output a probability value for each node in the output layer, and the probability values sum up to 1.0. The probabilities are converted to encoded class labels after applying a threshold. This will give an output of (1,0,0), (0,1,0) or (0,0,1) to classify the composition as liquid, gas or two-phase. The output from the classification model is used as part of the input into the regression model, but only two of the three output phase classes are used to avoid the dummy variable trap (Suits, 1957) during the training of the regression model.

Table 3. Summary of the neural network used for phase classification.

Dataset	Training: 700,000 Testing: 150,000 Validation: 150,000
Neurons per layer	Input: 4 neurons, Hidden layers: 128 neurons each. Output layer: 3 neurons.
Batch size	256
Number of epochs	250
Optimizer	Adam
Loss function	Categorical Cross-entropy
The hidden layer activation function	ReLU
Output layer activation function	SoftMax
Metric	Accuracy

5.3.2 Implementation of the regression models.

A standard DNN regression model and a PINN regression model are created with similar architecture as summarized in Table 4. The results of the predictive performance of the two models are compared and discussed in Chapter 6 to understand the effect of incorporating physics into the second step using the PINN.

I used the SciANN Python package (Haghighat & Juanes, 2021) for the PINN regression model to take advantage of the flexibility of turning the physics constraints on or off. The SciANN package is based on TensorFlow (TensorFlow, 2019), so I had access to all the deep learning functionalities of this robust deep learning package. For the standard DNN model, I implemented a fully connected feed-forward deep neural network with the loss function given in Equation [[47]. For the PINN model, the standard loss function is modified to include the physics constraints as shown in Equation [46]. Table 4 summarizes the model parameters used in the DNN and PINN models.

Table 4. Summary of the Model Parameters for the DNN and PINN models

	PINN	DNN
Dataset	Training: 700,000 Testing: 150,000 Validation: 150,000	Training: 7,000,000 Testing: 150,000 Validation: 150,000
Network	Input layer: 6 neurons, Hidden layers: 128 neurons each Output layer: 7 neurons	Input layer: 6 neurons, Hidden layers: 128 neurons each Output layer: 7 neurons
Batch size	256	256
Number of epochs	300	300
Optimizer	Adam	Adam
Modified Loss function	$\lambda_1 MSE_1 + \lambda_2 MSE_2 + \lambda_3 MSE_3$	MSE_1
The hidden layer activation function	ReLU	ReLU
Output layer activation function	Sigmoid	Sigmoid
Metrics	RMSE, R2	RMSE, R2

5.4. K-fold cross-validation

Given the stochastic nature of the random initial weights and gradient optimizers used to train the ML model, the predicted outputs frequently vary even when the same model is trained multiple times on the same input data. To alleviate this problem in this work, seven-fold cross-validation is performed to obtain a robust estimate of the model's performance on unseen data, using the mean and standard deviation of the adjusted coefficient of determination. Figure 5.4 provides a schematic illustration of the seven-fold cross-validation. On different subsets of the training data, seven models with the same parameters are trained and evaluated. Six-sevenths of the combined training and validation data are used to train the model in each of the seven-folds, with the remaining one-seventh utilized to compute the model's performance. As a result, seven unique estimates for each of the model parameters are obtained. The box plots shown in the following chapter are based on these seven estimates.

Additionally, the model averaging or bootstrap aggregating ("bagging" for short) technique is used to reduce the generalization error (Breiman, 1996). In this technique, the seven models generated for each combination of model parameters are saved and combined to form an ensemble model. This is achieved by simply weighting the model predictions from each model equally. Bagging is well-known to outperform the single best-performing model because of the reduction in the generalization error (Goodfellow et al., 2016).

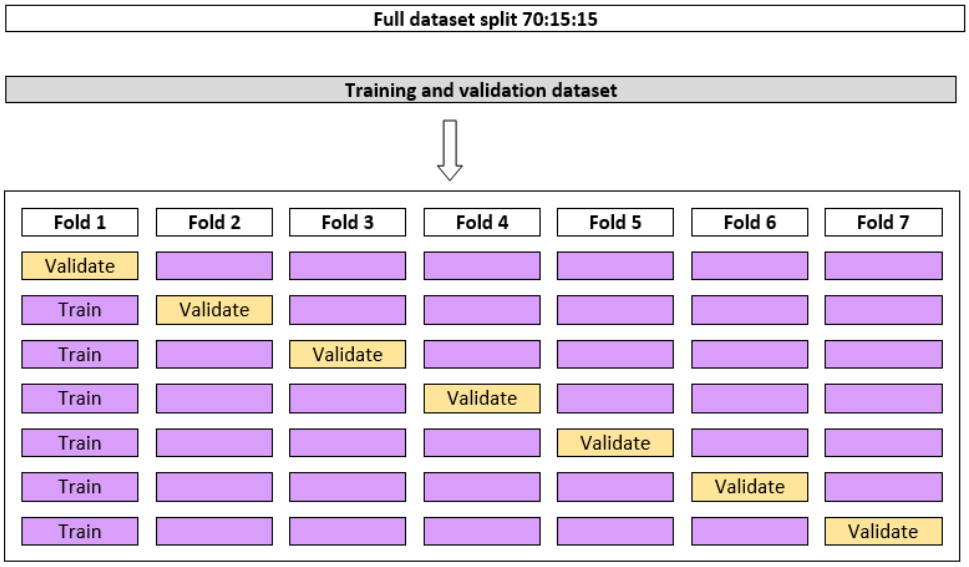

Figure 5.4. Seven-fold cross-validation

6. RESULTS AND DISCUSSION

This chapter starts with a discussion of the results obtained from using a DNN classification model to determine the phase of the fluid mixture based on the given overall mole fractions and pressure. To evaluate the incorporation of physics via the PINN model, we compare the PINN model results to those from a standard DNN that does not incorporate the thermodynamics constraints explicitly. Finally, we compare the phase diagrams based on the PINN model predictions of the overall mole fractions to those based on a standard DNN model, a linear regression model, and the phase envelopes computed using the actual overall mole fractions in the test dataset.

6.1. Discussion of results for phase classification

This section presents the results of the phase classification, where we determine if the composition is in the single-phase liquid, gas, or two-phase state. It also discusses the results of training the DNN model on a dataset with an imbalanced class distribution and the effect on the model's predictive ability.

The model is trained to predict three output variables that classify the phase as a liquid, vapor, or two-phase for any given composition and pressure. The model took 94 minutes to fit the 700,000 training dataset and 5 seconds to predict the flash output variables using the 150,000 validation dataset. The results are achieved using a computer with an Intel ® Xeon ® Silver 4216 CPU 16-Core, 32-Thread @ 2.10GHz and a 64 GB RAM. The model's overall accuracy is 97.7%, and the value of the loss function is 0.0872. Figure 6.1 and Figure 6.2 present the model accuracy and loss function evolution during the training and validation.

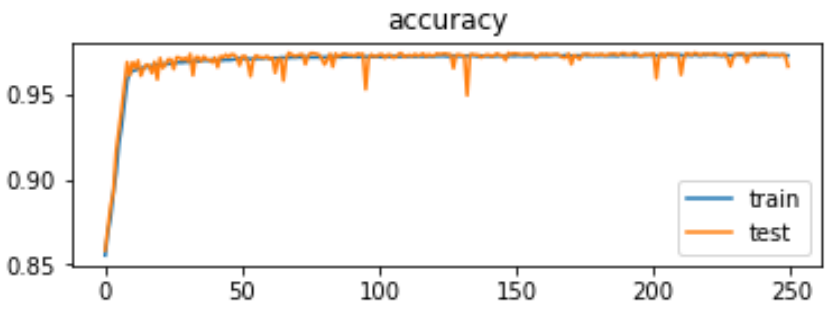

Figure 6.1. Overall accuracy of the classification model

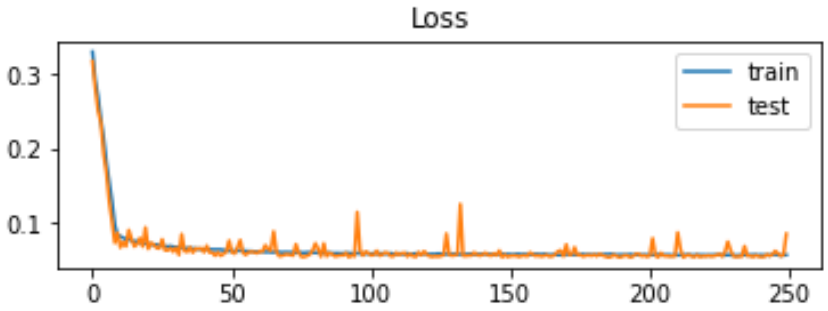

Figure 6.2. Loss during the training and validation of classification model

Accuracy is defined as the ratio of correctly predicted observations to the total observations. From the results, we can see that the overall accuracy of the classification model is 97.7%. However, since the model predicts three different phases, it is crucial to measure the model's predictive ability for each phase separately. This is done by individually evaluating each phase's precision, recall, and F1-score. Precision is the ratio of correctly classified positive observations to the total number of observations classified as positive, i.e. (TP/(TP+FP)). High precision indicates an observation labeled as positive is indeed positive (a small number of FP). The recall, also known as sensitivity, is the ratio of correctly classified positive observations to all observations classified as positive, i.e. (TP/(FP+FN)). The F1 score is the weighted average of the recall and precision, i.e. ((2*precision*recall)/ (precision + recall)).

Figure 6.3 shows the confusion matrix for the gas phase. From the matrix, we can see that the model correctly predicted the classification of 266 fluid mixtures as gas phase, and it wrongly predicted 318 fluid mixtures as not gas phase when in fact, they are. We also computed the precision, recall, and F-1 score for the single-phase gas state as 78.2%, 45.5%, and 57.6%, respectively. These metrics indicate that although the overall model accuracy is very high (97.7%), its ability to correctly classify fluid mixtures in the single-phase gas state is weak. This is typically due to the imbalance or uneven class distribution in the population of fluid mixtures used to train the deep learning model. This issue is addressed at the end of this section.

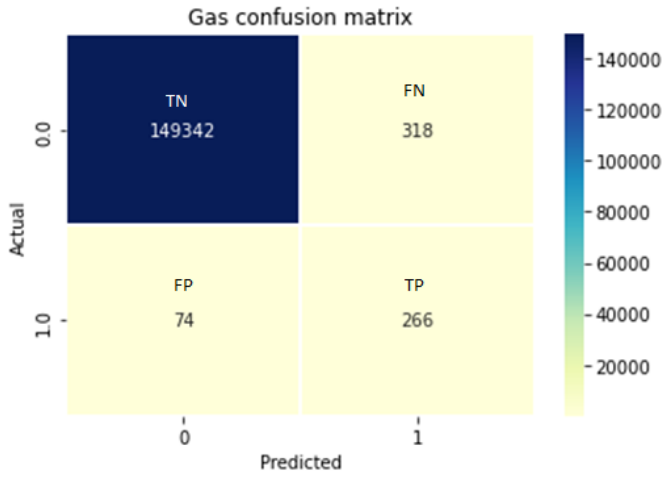

Figure 6.3. Gas Confusion Matrix

Figure 6.4 shows the confusion matrix obtained for the liquid phase. The model correctly classified 111,604 fluid mixtures as liquid phase, and it incorrectly predicted that 2954 fluid mixtures are not in the liquid phase when in fact, they are. To evaluate the model's ability to correctly predict the classification of the fluid mixtures in the liquid phase, we computed the recall (98%), precision (97%), and F-1 score (98%). These results indicate that the model is able to correctly classify the fluid mixtures in the liquid phase.

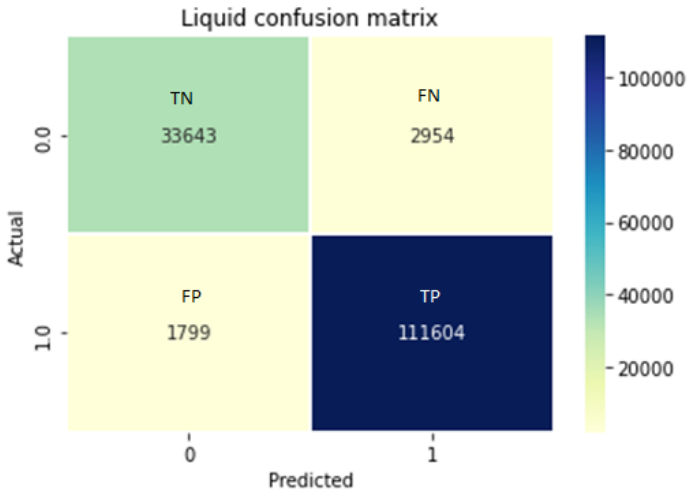

Figure 6.4. Liquid Phase Confusion Matrix

Figure 6.5 shows the confusion matrix for the two-phase state. The confusion matrix for the two-phase is shown in Figure 6.5. We can see from the matrix that the model correctly classified 33,030 fluid mixes as two-phase while incorrectly predicting 1795 fluid mixtures as not two-phase when they are. As previously done for the liquid and gas phase, to assess the model's ability to forecast the classification of fluid mixes in the two-phase correctly, we compute the recall, precision, and F-1 score, which are 91%, 95%, and 93%, respectively. Comparing these values to the corresponding values for the liquid state, we can conclude that the model's ability to correctly classify fluid mixtures in the two-phase state is slightly less than its ability to classify fluix mixtures in the single-phase liquid state.

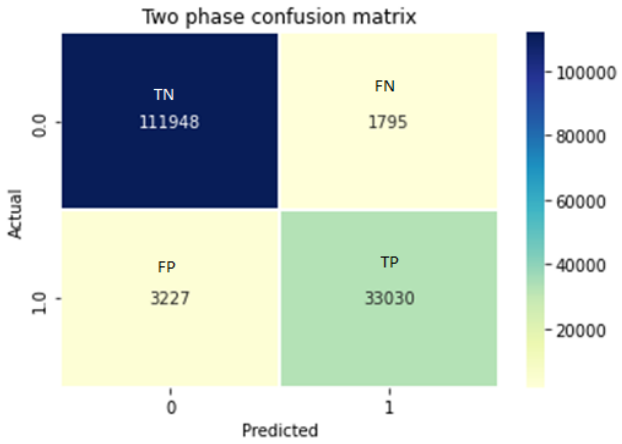

Figure 6.5. Two-phase Confusion Matrix

Table 5 summarizes the precision, recall, and F-1 scores of the three phases in the dataset with the imbalanced class distribution. Although the model has a high overall accuracy of 97.7%, its low values of precision, recall, and F1 score in the gas phase indicate that it will not correctly classify the gas phase within an acceptable range of error.

Table 5. Summary statistics for the dataset with an imbalanced class distribution

	Precision	**Recall**	**F1-score**
Gas	0.78	0.46	0.58
Liquid	0.97	0.98	0.98
Two-phase	0.95	0.91	0.93

The validation dataset had 150,000 observations with a class distribution of 76.4%, 23.2%, 0.397% of the data in the liquid, two-phase, and gas states, respectively. The number of observations in each phase of these states was 114558, 34825, and 584 for the liquid, two-phase, and gas states, respectively. From these numbers, it is apparent that the dataset has an imbalanced or highly skewed class distribution. The measures of accuracy for training data with imbalanced class distribution are typically dependent on the number of observations that the model is trained on for

each phase. This is the reason that the precision, recall, and F-1 score values were lowest in the single-phase gas state and highest in the single-phase liquid state.

To resolve this limitation, we modificed the dataset to include more fluid mixtures in the gas state. A new deep learning model was then trained with the more evenly balanced dataset. Figure 6.6 shows the overall accuracy of a classification model trained on a dataset with an even distribution of fluid mixtures in the liquid, gas, and two-phase states. The model took 60 minutes to fit the 700,000 fluid mixtures in the training dataset and 32 seconds to predict on the 150,000 fluid mixtures in the validation dataset. The model's overall accuracy is 98.2%, and the value of the loss function is 0.0427.

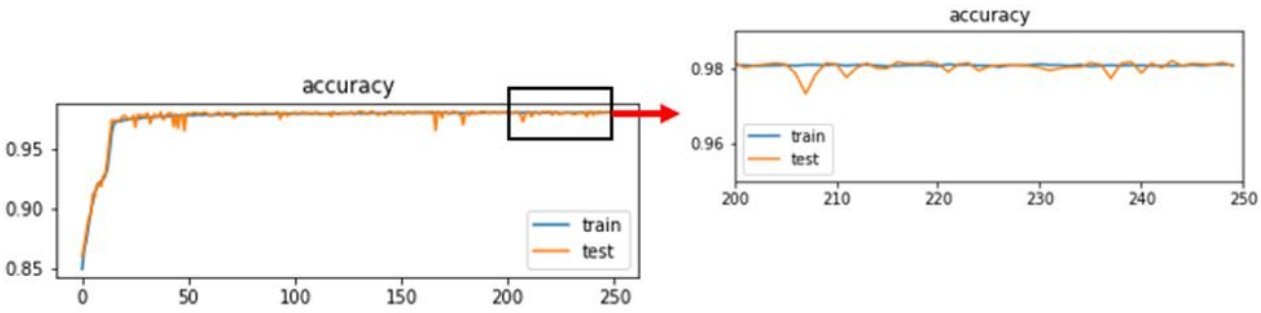

Figure 6.6. Accuracy of an evenly distributed dataset

Figure 6.7 shows that the training and validation loss function value decreases to a minimal value, indicating a good fit for the model with a balanced dataset. The minimal difference between the training and validation data loss functions indicates that the trained model does not overfit the training data provided.

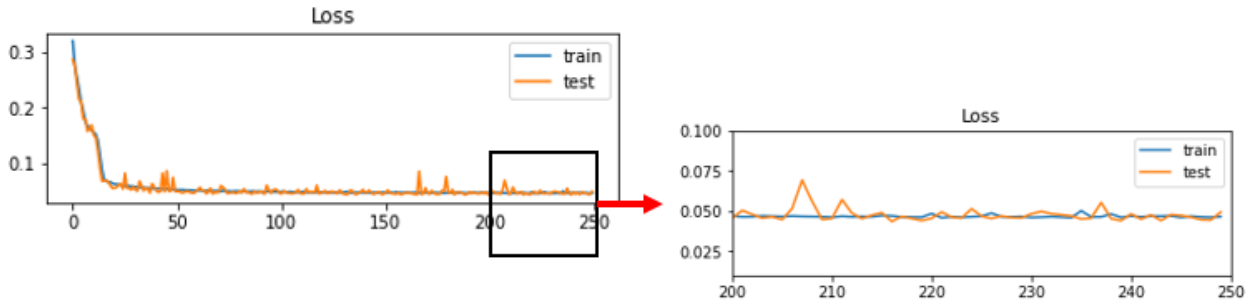

Figure 6.7. Loss propagation of an evenly distributed dataset

Figure 6.8 shows the gas phase confusion matrix generated from a model trained with a balanced class distribution dataset. From the gas confusion matrix, the recall is computed as 99.4%, the precision is 99.8%, and the F-1 score is 99.6%.

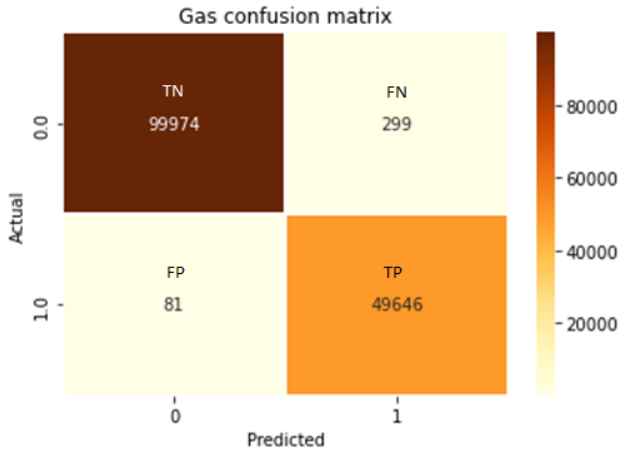

Figure 6.8. Gas confusion matrix for evenly distributed dataset

The confusion matrix for the fluid mixtures in the liquid phase predicted by a model trained and evaluated on a dataset with a balanced class distribution is shown in Figure 6.9. The recall is 98.9% from the matrix, the precision is 95.7%, and the F-1 score is 97.3%.

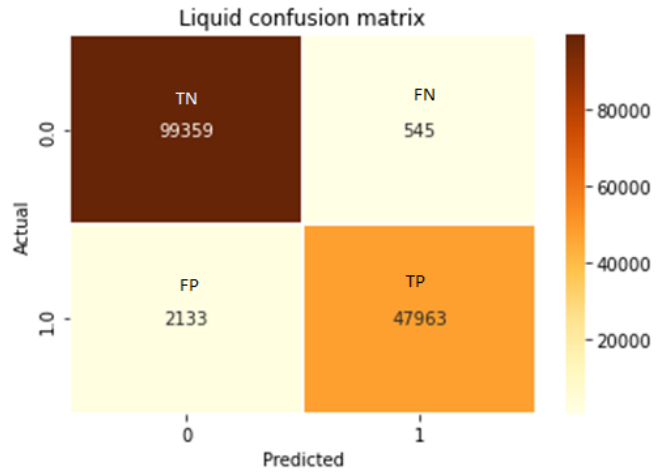

Figure 6.9. Liquid confusion matrix for evenly distributed dataset

Figure 6.10 shows the two-phase confusion matrix for fluid mixtures classified using a model trained and evaluated on a dataset with a balanced class distribution. From Figure 6.10, we can calculate the recall, precision, and F-1 score for the two-phase class as 96%, 98.5%, and 97.2%, respectively.

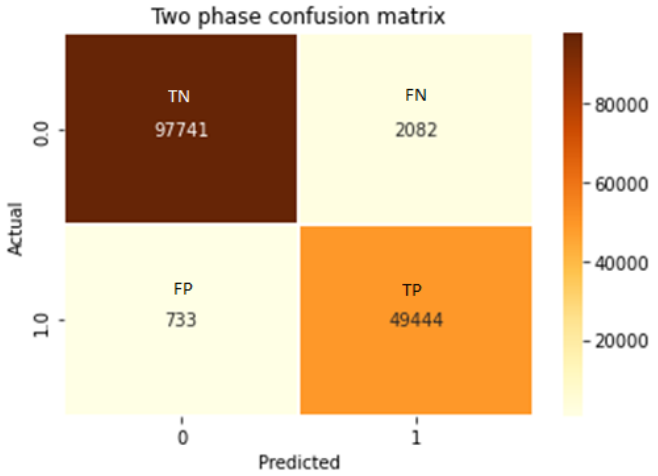

Figure 6.10. Two-phase confusion matrix for evenly distributed dataset

Table 6 summarizes the precision, recall, and F-1 score of the three phases in a dataset with a balanced class distribution. We observe that the classification model's overall accuracy increased from 97% to 98% from these results. In the case of the evenly distributed dataset, the model's overall accuracy reliably represents the model's predictive ability for all three phases within an acceptable range of error.

Table 6. Summary statistics for the dataset with a balanced class distribution

	precision	**recall**	**F1-score**
Gas	0.994	0.998	0.996
Liquid	0.989	0.957	0.973
two-phase	0.960	0.985	0.972

These results indicate that for compositional fluid property modeling using machine learning, it is crucial to ensure that the dataset generated from the traditional two-phase flash process has an approximately equal number of observations from each phase before it is used to train and evaluate the deep learning models.

6.2. Discussion of results for Regression using DNNs and PINNs

This section discusses the results of training a standard DNN and a PINN model to predict the phase mole fractions and vapor fraction, given the overall composition, pressure, and temperature. To quantify the significance of incorporating the physics constraints, this section discusses the standard DNN model trained to provide a reference base case. The interphase mass balance and component balance constraints are then incorporated and optimized one at a time to clearly distinguish the effect of incorporating the constraints and keep the optimization simple to

implement and interpret. This section is divided into three parts. The first discusses the DNN model, while the remaining two subsections discuss the results obtained from the PINN model with the two physics constraints.

6.2.1 Regression using the DNN model.

This subsection discusses the results from the standard DNN model. The loss function in the standard DNN model is the MSE. This loss only measures the error associated with the misfit of the data. Using equation [46], the loss function for the DNN model is obtained by setting the values of λ_2 and λ_3 to zero and setting λ_1 to one. Therefore equation [46] is simplified to represent the loss function equation for the DNN model as:

$$L = MSE_1 \quad [55]$$

[5]

The metrics used to evaluate the performance of the DNN model are summarized in Table 7. These results will later be compared to the results from the PINN models to understand the effect of incorporating thermodynamics constraints on model performance.

Table 7. Results of the DNN model.

	DNN
Overall Model R^2	0.9663
Overall RMSE	0.0399
Loss	0.0112

Figure 6.11 shows the loss learning curves of the DNN model for the training and validation dataset. The results indicate that the training and validation loss functions reduce to a very low value at the end of the training. The close match between the training and validation losses indicates that the trained model does not overfit the input data.

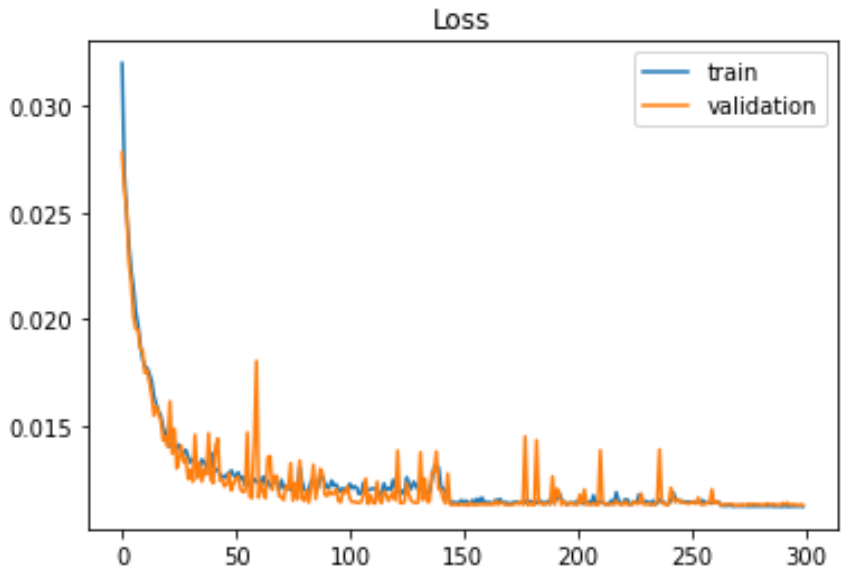

Figure 6.11. The learning curve for the DNN model.

6.2.2 PINNs with inter-phase mass balance constraint

Next, we discuss the results from incorporating the inter-phase mass balance constraint into the deep learning process and compare the results of the PINN model with the interphase mass balance constraint to those from the standard DNN model. Using the weighted sum of MSEs, we obtain the results of the PINN model with only the interphase mass balance constraint by simply setting λ_3 to zero, λ_1 to one, and assigning different values to λ_2. So, the loss function equation (equation [46]) simplifies to the following:

$$L = MSE_1 + \lambda_2 MSE_2 \quad [56]$$

Considering that the goal is to minimize the loss function L, it is essential to determine the optimum value of λ_2 at which the model minimizes the errors associated with the data misfit (MSE_1) and the interphase mass balance (MSE_2) (Ihunde & Olorode, 2022). To determine the optimum value of λ_2, the RMSE of the interphase mass balance constraint and the coefficient of determination (R^2) are plotted against weight λ_2 to find the optimum value of λ_2 where the RMSE of the

interphase mass balance constraint is minimized, and the R^2 is still very high (Ihunde & Olorode, 2022).

Figure 6.12 shows the box plot created using the seven RMSE and R^2 values obtained from the seven-fold cross-validation results plotted against weights λ_2 . The RMSE measures the degree to which the predictive model honors the laws governing phase equilibrium, and the R^2 quantifies the model's overall accuracy (Ihunde & Olorode, 2022). The box plot provides insight into the model's performance as the weights gradually increase and facilitates the determination of the optimal weight for the inter-phase mass balance constraint. It is worth noting that we initially attempted a computationally intensive multi-objective optimization of the weights however, gradually increasing the weights while monitoring the evaluation metrics provides insights into the model's performance at various weight values (Ihunde & Olorode, 2022). So, it was deemed a more pragmatic and useful approach.

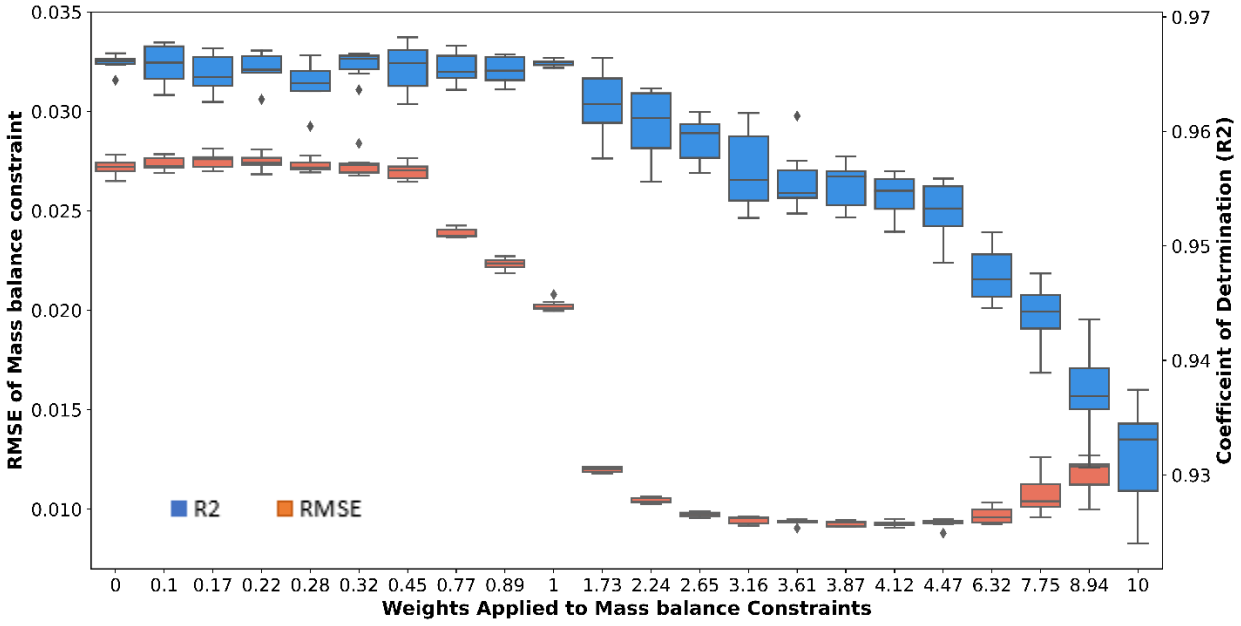

Figure 6.12. Box plots show how the model accuracy and physics constraint errors (indicated by the R^2 and RMSE, respectively) vary with the weights applied to the interphase mass balance constraint.

In Figure 6.12, we can see that the weights (λ_2) are set to vary from 0 to 10. The weight of zero is the reference weight that represents the results from the standard DNN model, where the model does not impose any physical constraints. So, equation [56] simplifies into equation [55] when $\lambda_2 = 0$. When λ_2 is increased from 0 to 2, the RMSE of the interphase mass balance error decreases from 0.0027 to 0.0012. Beyond the weight of 2, the R^2 value decreases appreciably. So, we can conclude from Figure 6.12 that the optimal weight for the interphase mass balance constraint is ~1.73. This weight is selected because the RMSE is only marginally reduced when the weight increases above the weight of 1.73, but the model's R^2 decreases significantly.

Table 8 shows a comparison between the PINN model with the interphase mass balance constraint and the DNN model without the interphase mass balance constraint using the R^2 and RMSE values (Ihunde & Olorode, 2022). This table presents the best single PINN and DNN models rather than the ensemble model results. When benchmarking models, utilizing a single model rather of an ensemble model helps to avoid the natural effect of model averaging-induced model performance improvement. (Goodfellow et al., 2016). From Table 8, the RMSE for the interphase mass balance constraint is 55% lower in the PINN model than in the DNN model, resulting in a lower total RMSE for the PINN model. The R^2 for the PINN is approximately the same as in the DNN model. This indicates that incorporating the interphase mass balance constraint results in a model that honors the physical constraint without decreasing the model's accuracy.

Table 8. Comparison of DNN to PINN with interphase mass balance constraint

	PINN	**DNN**
Overall Model R^2	0.9658	0.9663
Overall RMSE	0.0356	0.0399
$RMSE_2$	0.0118	0.0265

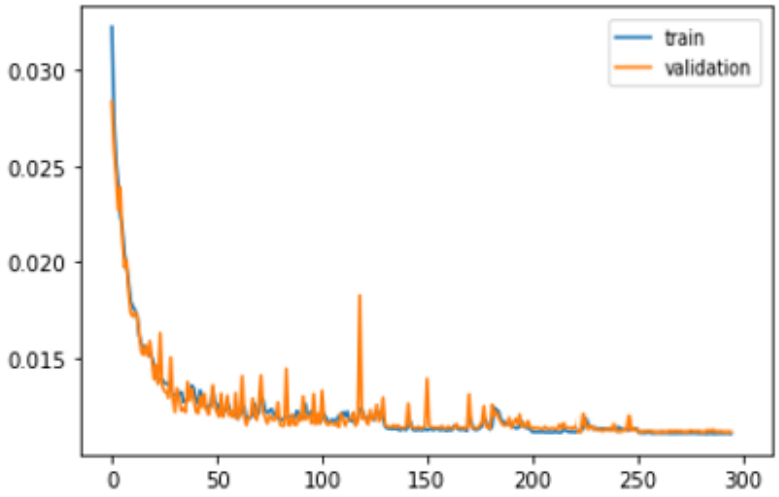

Figure 6.13. Progression of the loss function for the inter-phase mass balance constraint at a weight of 1.73

Figure 6.13 shows the training progression of the modified loss function for the best PINN model with the inter-phase mass balance constraint at the optimal weight of 1.73. The overall RMSE of the PINN model with the inter-phase mass balance constraint is 0.0356. A closer look at the loss in Figure 6.13 shows that the model does not overfit the data because the training and validation loss functions decrease to a small value with a minimal gap between the two loss function values at the end of the training and validation.

6.2.3 PINNs With Component Balance Constraint

This subsection discusses the outcomes of training PINNs with a component balance constraint. As in the Section 6.2.2, I use a weighted sum of MSEs to obtain the results of the PINN model with the component balance constraint by setting the value of λ_2 to zero, λ_1 to one, and assigning different values to λ_3(Ihunde & Olorode, 2022). For the component balance constraint, the loss function given in equation [46] is simplified as follows (Ihunde & Olorode, 2022):

$$L = MSE_1 + \lambda_3 MSE_3 \quad [57]$$

Figure 6.14 shows the RMSE of the component balance constraint and the R^2 against the weight λ_3 (Ihunde & Olorode, 2022). This box plot was created using the seven RMSE and R^2 values obtained from the seven-fold cross-validation results (Ihunde & Olorode, 2022). It provides insight into the model's performance as the weights gradually increase and helps determine the optimal weight for the component balance constraint. The RMSE measures the degree to which the predictive model adheres to the laws governing phase equilibrium, and the R^2 quantifies the model's overall accuracy (Ihunde & Olorode, 2022). Like in Figure 6.12, the reference point for the standard DNN model is obtained when λ_3 is set to zero, which implies that no physical constraints enforced. The RMSE decreases from 0.15% to 0.02% while the R^2 remains relatively constant as the value of λ_3 increases from zero to 2.24 (Ihunde & Olorode, 2022). Beyond a weight of 2.24, the RMSE does not decrease any further, and the R^2 begins to decline. Therefore, the optimal weight for the component balance constraint is ~2.24 (Ihunde & Olorode, 2022).

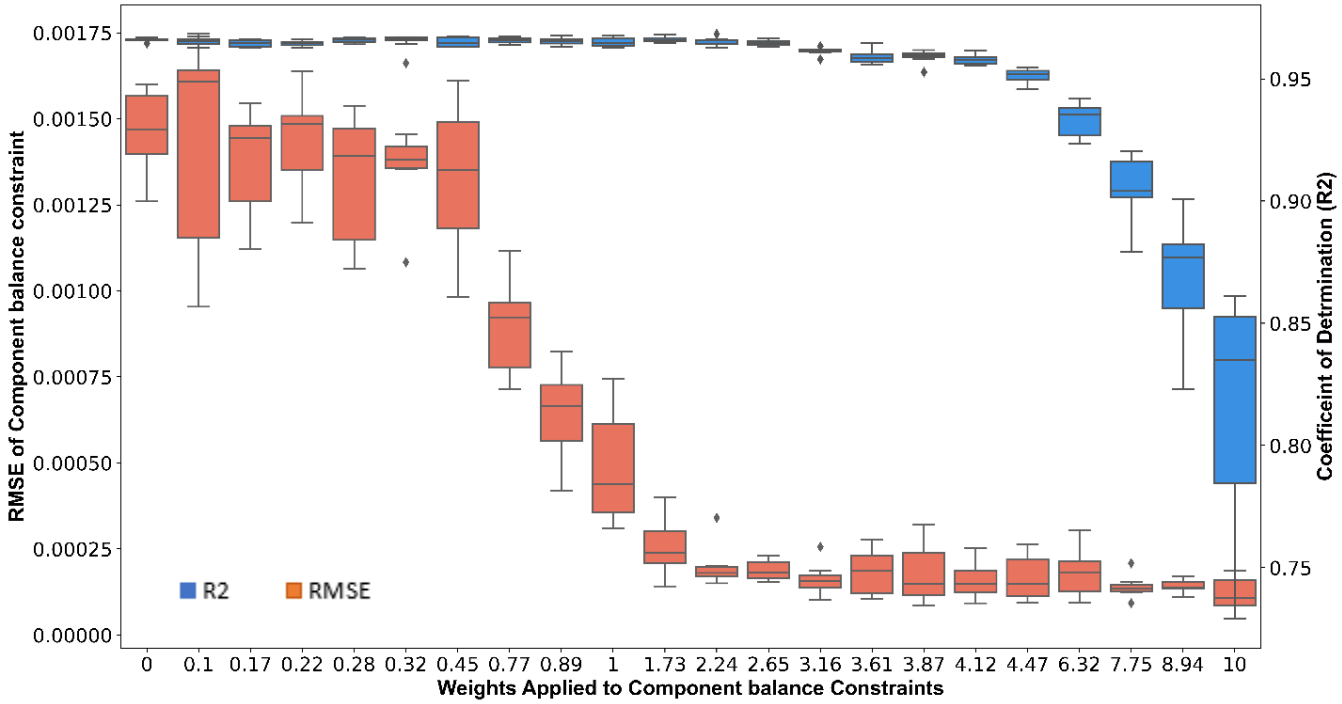

Figure 6.14. Box plots show how the model accuracy and physics constraint errors (indicated by the R^2 and RMSE, respectively) vary with the weights applied to the component balance constraint.

Table 9 compares the PINN and DNN models using the R^2 and the RMSE of the data misfit and the component balance constraint (Ihunde & Olorode, 2022). As in Table 7, this table compares single best models rather than ensemble models. It shows that the RMSE for the component balance constraint is 88% lower in the PINN model than in the DNN model (Ihunde & Olorode, 2022). This indicates that incorporating the component balance constraint results in a model that honors the physical constraint without a decrease in the model's accuracy at the selected λ_3 value of 2.24.

Table 9. Comparison of DNN to PINN with component balance constraint

	PINN	**DNN**
Overall Model R^2	0.9683	0.9663
Overall RMSE	0.03887	0.0399
$RMSE_3$	0.00015	0.00126

Figure 6.15 shows the training progression of the modified loss function for the best PINN model with the component balance constraint at the optimal weight of 2.24. The overall RMSE of the PINN model with the component balance constraint is 0.0388. Figure 6.15 shows that the training and validation loss functions decrease to a minimal value and the difference between them is negligible at the end of the training process. This indicates that the resulting model does not overfit the training data.

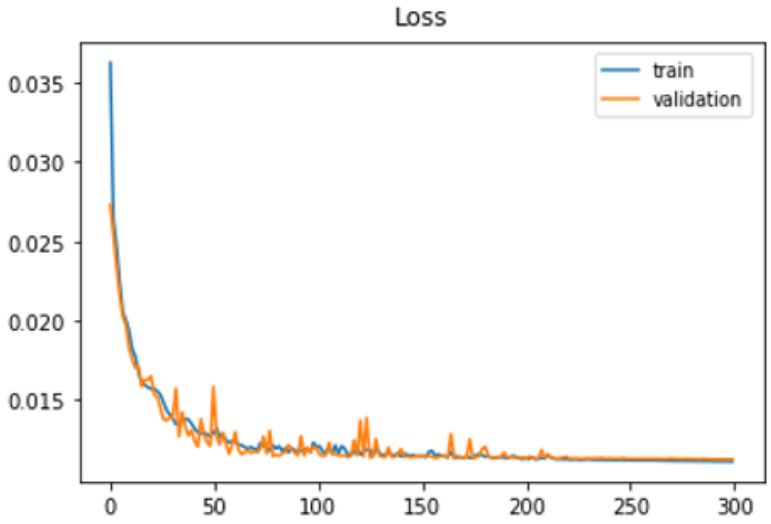

Figure 6.15. Evolution of the loss function for the component balance constraint at a weight of 2.24

A comparison of the RMSEs in Figure 6.12 and Figure 6.13, as well as Table 8 and Table 9, shows that the RMSEs of the component balance constraint are one order of magnitude smaller than those of the interphase mass balance constraint (Ihunde & Olorode, 2022This may be due to the relative ease with which the deep learning model can learn the component balance constraint without explicitly implementing it into the loss function (Ihunde & Olorode, 2022). In contrast, the interphase mass balance constraint is a linear combination of the vapor fraction, overall, and phase

mole fractions (Ihunde & Olorode, 2022). This is more complex than the simple summation of mole fractions in the component balance constraint.

Additionally, the component balance error could be much smaller because the standard DNN models can infer that the phase compositions should sum to one based on the implicit component balance in the overall mole fractions provided in the training data (Ihunde & Olorode, 2022). From these results, we can infer that the inter-phase mass balance constraint appears to be more important than the component balance constraint. So, the comparison of the model results in the next section is based on the PINN model with the inter-phase mass balance constraint.

6.3. Comparison of model results

This section compares the results of the PINN model predictions to those from standard DNN and linear regression models using phase envelopes. To make this comparison, we use the PINN, DNN, and linear regression models to predict the phase mole fractions x_i, y_i, and volume fraction V of the 150,000 fluid mixtures in the test dataset. The interphase mass balance errors for each of the predicted fluid mixtures in the PINN model are then used to rank the fluid mixtures in percentiles. Using equation [21], the respective overall mole concentrations z_i of the fluid mixtures at the 99th, 75th, 50th, 25^{th,} and 1st error percentiles are computed from the predicted x_i, y_i, and V for each of the models.

Table 10**Error! Reference source not found.** shows each fluid mixture's total and interphase mass balance errors at the specified percentiles (Ihunde & Olorode, 2022). The total error is the sum of the absolute difference between the x_i, y_i, and V values predicted by the PINN and DNN models and their corresponding values in the test data (Ihunde & Olorode, 2022). The DNN and PINN interphase error is the absolute value of "$z_i - Vy_i - (1 - V)x_i$" from equation [21] (Ihunde &

Olorode, 2022). The results in the table indicate that the PINN model outperforms the DNN model at all percentiles (Ihunde & Olorode, 2022).

Table 10. Summary of the performance of the DNN and PINN models at specific percentiles

Percentile	DNN Total error	DNN Interphase error	PINN Total error	PINN Interphase error
99	1.20E-04	7.00E-05	8.60E-05	2.00E-06
75	3.91E-03	7.30E-04	1.33E-03	1.20E-05
50	2.37E-02	2.17E-02	1.22E-02	2.16E-03
25	1.35E-01	2.19E-02	7.37E-02	1.47E-02
1	4.13E-01	2.83E-02	4.07E-01	1.34E-02

Using the computed overall mole fractions as inputs into CMG's Winprop (Computer Modelling Group Ltd, 2017), we generate the pressure-temperature (P-T) phase diagrams shown in Figure 6.16. Figure 6.16(a) - Figure 6.16(e) show the phase envelopes generated from the predictions of z_i by the PINN model (solid blue line), DNN model (solid green lines), linear regression model (solid orange lines), and the actual z_i values in the test data (dotted red lines) at the 1^{th}, 25^{th}, 50^{th}, $75^{th,}$ and 99^{st} percentiles of the error. The actual phase envelopes (dotted red lines) were obtained from the test data's overall composition (z_i) values for the corresponding fluid mixtures at the respective percentiles. Although these phase diagrams cover a wide range of temperature and pressure values, the input data provided as input into Winprop corresponds to a single point in the P-T phase diagram (Ihunde & Olorode, 2022). So, it is unrealistic to expect a neural network model trained to predict the fluid properties for a distinct fluid mixture at only one pressure and temperature to match the actual phase behavior over the wide pressure and temperature range over which it is not trained (Ihunde & Olorode, 2022).

Figure 6.16(a) shows phase diagrams based on model predictions of the overall mole fractions for a fluid mixture at pressure and temperature values where it exists in the single-phase liquid state

(Ihunde & Olorode, 2022). In this case, the three estimated z_i values for C_1, C_2, and C_3 exactly match the actual z_i values because the x_i's are identical to the $z_i$values, whereas y_i and V are zeros (Ihunde & Olorode, 2022). Figure 6.16(b) to (e) shows the P-T phase diagrams for the two-phase fluid mixtures at the specified percentiles (Ihunde & Olorode, 2022). The results show that the phase diagrams from the PINN model predictions outperform the standard DNN model and linear regression model predictions in all the two-phase fluid mixture cases. The trend in these phase envelopes is more apparent at the isothermal temperature of 176 ^{o}F at which all the fluid compositions were specified (Ihunde & Olorode, 2022). At this temperature, the figure shows that as the error percentile decreases, the deviation of the models from the actual saturation pressure increases as expected (Ihunde & Olorode, 2022).

Although the R^2 and RMSE values for the models in Table 8 do not appear to indicate a statistically significant difference in the results, the phase diagrams indicate that incorporating physics with PINNs results in predictions that more accurately describe the phase behavior of compositional fluid mixtures (Ihunde & Olorode, 2022). In spite of strong performance metrics, DNN models may not adhere to thermodynamics constraints of phase equilibrium since they are not applied during the deep learning process. By adding the physics restrictions via custom loss functions, PINNs address this shortcoming and produce better model predictions that honor the physical limits of phase equilibrium. These customized loss functions serve as a form of physics-based regularization that aids in the solution of ill-posed problems. (Kashinath et al., 2021). Finally, it is worth noting that our performance analysis of the trained PINN model shows that it is 145 times faster than the standard two-phase flash algorithm (Ihunde & Olorode, 2022).

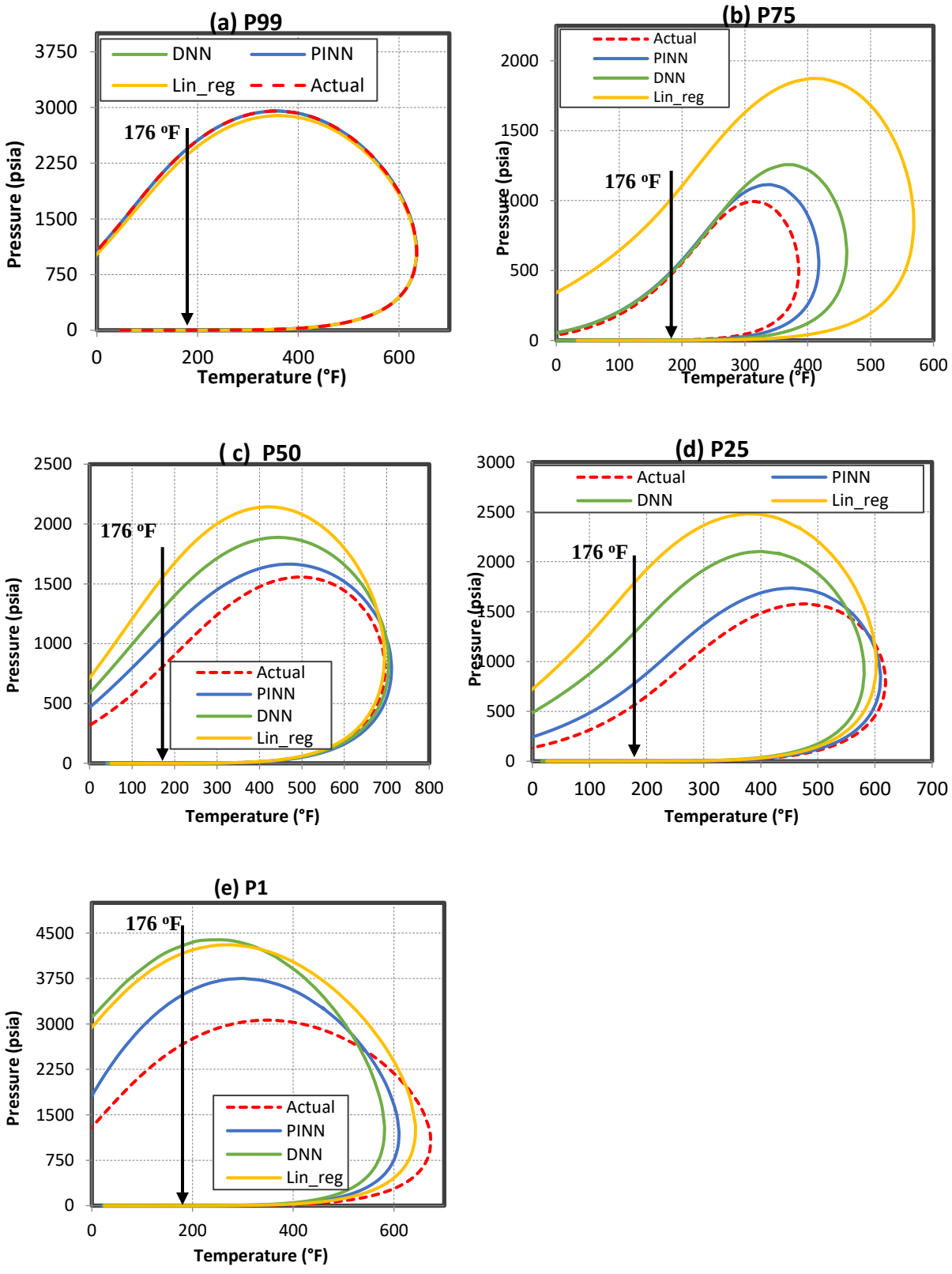

Figure 6.16. Phase envelopes for compositions at different percentiles indicate that the PINN model yields a better description of the phase behavior than the DNN

7. CONCLUSIONS

This work demonstrates how thermodynamics constraints can be incorporated into neural network models trained on data from two-phase flash calculations. The data in this work has one million unique fluid mixtures generated using a space-filling mixture design. The flash output variables x_i, y_i, and V are predicted from the overall composition, pressure, and temperature of the fluid mixtures. During the training of the deep learning models, classification is first performed to identify the phase of each fluid mixture then the regression is performed to predict the flash output variables. The main takeaway from the classification using the deep learning model is that it is vital that the dataset used to train the classification model has a balanced class distribution for the phases. If the dataset has an imbalanced class distribution, the metrics used to evaluate the model's performance will not represent the model's predictive ability for each phase. Instead, it will be skewed by the phase with the largest number of fluid mixtures.

After identifying the phases of the fluid mixtures, regression models are trained to predict the flash output variables x_i, y_i, and V from the overall composition, pressure, and temperature of the fluid mixtures. This work shows how the thermodynamic constraints are incorporated by replacing the standard loss function with a weighted sum of the standard DNN MSE and the MSEs associated with thermodynamic constraints. Using the weighted sum of MSEs allows us to perform a sensitivity study to systematically determine the optimal weights to incorporate the physics constraints without compromising model accuracy.

The results show that the PINN model yields phase equilibrium predictions with over 55% lower physics constraint errors (RMSEs) when compared to the standard DNN model predictions. These lower physics constraint errors were achieved at little or no loss of overall model accuracy, as

evidenced by the insignificant (<0.05%) difference in the overall R^2 value of the PINN and DNN models. From the results of this work, we conclude that PINNs yield model predictions that honor physical constraints without lowering overall model accuracy and are 145 times faster than the standard two-phase flash calculation procedure.

Comparing the phase diagrams generated using the output from PINNs, DNNs, and linear regression models to those estimated using the test data indicates the importance of incorporating physics into DNN models for two-phase flash. The results show that although the PINN and DNN models have nearly the same R^2, the phase diagrams show that the PINN model outperforms the DNN model in predicting the phase behavior of these fluid mixtures (Ihunde & Olorode, 2022).

8. REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., . . . Isard, M. (2016). Tensorflow: A system for large-scale machine learning. 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16),

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. *Journal of machine learning research, 18*.

Belkadi, A., Yan, W., Michelsen, M. L., & Stenby, E. H. (2011). Comparison of two methods for speeding up flash calculations in compositional simulations. SPE Reservoir Simulation Symposium,

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., . . . Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. Proceedings of the Python for scientific computing conference (SciPy),

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123-140.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., . . . Zhang, Z. (2015). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*.

Coats, K. H. (1980). An equation of state compositional model. *Society of Petroleum Engineers Journal*, *20*(05), 363-376.

Computer Modelling Group Ltd. (2017). *Winprop User Guide*. In (Version 2017) Computer Modelling Group Ltd.

Daw, A., Thomas, R. Q., Carey, C. C., Read, J. S., Appling, A. P., & Karpatne, A. (2020). Physics-guided architecture (pga) of neural networks for quantifying uncertainty in lake temperature modeling. Proceedings of the 2020 siam international conference on data mining,

Dourado, A., & Viana, F. A. (2020). Physics-informed neural networks for missing physics estimation in cumulative damage models: a case study in corrosion fatigue. *Journal of Computing and Information Science in Engineering*, *20*(6), 061007.

Firoozabadi, A. (2016). *Thermodynamics and applications in hydrocarbon energy production*. McGraw-Hill Education.

Firoozabadi, A., & Pan, H. (2000). Fast and robust algorithm for compositional modeling: Part i-stability analysis testing. SPE Annual Technical Conference and Exhibition,

Fraces, C. G., Papaioannou, A., & Tchelepi, H. (2020). Physics Informed Deep Learning for Transport in Porous Media. Buckley Leverett Problem. *arXiv preprint arXiv:2001.05172*.

Fuks, O., & Tchelepi, H. A. (2020). Limitations of physics informed machine learning for nonlinear two-phase transport in porous media. *Journal of Machine Learning for Modeling and Computing*, *1*(1).

Furukawa, H., Shoham, O., & Brill, J. (1986). Predicting compositional two-phase flow behavior in pipelines.

Gaganis, V., & Varotsis, N. (2012). Machine learning methods to speed up compositional reservoir simulation. SPE Europec/EAGE annual conference,

Gaganis, V., & Varotsis, N. (2014). An integrated approach for rapid phase behavior calculations in compositional modeling. *Journal of Petroleum Science and Engineering*, *118*, 74-87.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Gould, T. L. (1979). Compositional two-phase flow in pipelines. *Journal of Petroleum Technology*, *31*(03), 373-384.

Haghighat, E., & Juanes, R. (2021). Sciann: A keras/tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. *Computer Methods in Applied Mechanics and Engineering*, *373*, 113552.

Huang, D. Z., Xu, K., Farhat, C., & Darve, E. (2020). Learning constitutive relations from indirect observations using deep neural networks. *Journal of Computational Physics*, *416*, 109491.

Ihunde, T. A., & Olorode, O. (2022). Application of physics informed neural networks to compositional modeling. *Journal of Petroleum Science and Engineering*, 110175.

Joseph, V. R. (2016). Space-filling designs for computer experiments: A review. *Quality Engineering*, *28*(1), 28-35.

Joseph, V. R., Gul, E., & Ba, S. (2015). Maximum projection designs for computer experiments. *Biometrika*, *102*(2), 371-380.

Kashinath, A., Szulczewski, M. L., & Dogru, A. H. (2018). A fast algorithm for calculating isothermal phase behavior using machine learning. *Fluid Phase Equilibria*, *465*, 73-82.

Kashinath, K., Mustafa, M., Albert, A., Wu, J., Jiang, C., Esmaeilzadeh, S., . . . Singh, A. (2021). Physics-informed machine learning: case studies for weather and climate modelling. *Philosophical Transactions of the Royal Society A*, *379*(2194), 20200093.

Lekivetz, R., & Jones, B. (2015). Fast flexible space-filling designs for nonrectangular regions. *Quality and Reliability Engineering International*, *31*(5), 829-837.

Li, Y., Zhang, T., & Sun, S. (2019). Acceleration of the NVT flash calculation for multicomponent mixtures using deep neural network models. *Industrial & Engineering Chemistry Research*, *58*(27), 12312-12322.

Lie, K.-A. (2019). *An introduction to reservoir simulation using MATLAB/GNU Octave: User guide for the MATLAB Reservoir Simulation Toolbox (MRST)*. Cambridge University Press.

Manepalli, A., Albert, A., Rhoades, A., Feldman, D., & Jones, A. D. (2019). Emulating numeric hydroclimate models with physics-informed cGANs. AGU Fall Meeting 2019,

Michelsen, M. L. (1982a). The isothermal flash problem. Part I. Stability. *Fluid Phase Equilibria*, *9*(1), 1-19.

Michelsen, M. L. (1982b). The isothermal flash problem. Part II. Phase-split calculation. *Fluid Phase Equilibria*, *9*(1), 21-40.

MøYNER, O. (2021). *Advanced modeling with the MATLAB RESERVOIR SIMULATION TOOLBOX (MRST)*. Cambridge University Press. https://doi.org/10.1017/9781009019781

Nichita, D. V., Gomez, S., & Luna, E. (2002). Multiphase equilibria calculation by direct minimization of Gibbs free energy with a global optimization method. *Computers & chemical engineering*, *26*(12), 1703-1724.

Nichita, D. V., & Graciaa, A. (2011). A new reduction method for phase equilibrium calculations. *Fluid Phase Equilibria*, *302*(1-2), 226-233.

Okuno, R., Johns, R. T., & Sepehrnoori, K. (2010). A new algorithm for Rachford-Rice for multiphase compositional simulation. *SPE Journal*, *15*(02), 313-325.

Pal, N., & Mandal, A. (2021). Compositional simulation model and history-matching analysis of surfactant-polymer-nanoparticle (SPN) nanoemulsion assisted enhanced oil recovery. *Journal of the Taiwan Institute of Chemical Engineers*.

Pan, H., & Firoozabadi, A. (2001). Fast and robust algorithm for compositional modeling: part ii-two-phase flash computations. SPE Annual Technical Conference and Exhibition,

Peng, D.-Y., & Robinson, D. B. (1976). A new two-constant equation of state. *Industrial & Engineering Chemistry Fundamentals*, *15*(1), 59-64.

Rachford, H. H., & Rice, J. (1952). Procedure for use of electronic digital computers in calculating flash vaporization hydrocarbon equilibrium. *Journal of Petroleum Technology*, *4*(10), 19-13.

Raissi, M. (2018). Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, *19*(1), 932-955.

Raissi, M., & Karniadakis, G. E. (2018). Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, *357*, 125-141.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686-707.

Raissi, M., Yazdani, A., & Karniadakis, G. E. (2018). Hidden fluid mechanics: A Navier-Stokes informed deep learning framework for assimilating flow visualization data. *arXiv preprint arXiv:1808.04327*.

Russell, S. J., & Norvig, P. (2016). *Artifical intelligence: A modern approach*. Pearson.

SAS Institute Inc. (2020-2021). *JMP® 16 Design of Experiments Guide* https://www.jmp.com/content/dam/jmp/documents/en/support/jmp161/doe-guide.pdf

Suits, D. B. (1957). Use of dummy variables in regression equations. *Journal of the American statistical association*, *52*(280), 548-551.

TensorFlow. (2019). *TensorFlow*. https://www.tensorflow.org/

Trask, A. W. (2019). *Grokking deep learning*. Simon and Schuster.

Voskov, D. V., & Tchelepi, H. A. (2009). Tie-simplex based mathematical framework for thermodynamical equilibrium computation of mixtures with an arbitrary number of phases. *Fluid Phase Equilibria*, *283*(1-2), 1-11.

Wang, K., Luo, J., Wei, Y., Wu, K., Li, J., & Chen, Z. (2019). Artificial neural network assisted two-phase flash calculations in isothermal and thermal compositional simulations. *Fluid Phase Equilibria*, *486*, 59-79.

Wang, K., Luo, J., Wei, Y., Wu, K., Li, J., & Chen, Z. (2020). Practical application of machine learning on fast phase equilibrium calculations in compositional reservoir simulations. *Journal of Computational Physics*, *401*, 109013.

Wang, S., Sobecki, N., Ding, D., Zhu, L., & Wu, Y.-S. (2019). Accelerating and stabilizing the vapor-liquid equilibrium (VLE) calculation in compositional simulation of unconventional reservoirs using deep learning based flash calculation. *Fuel*, *253*, 209-219.

Ward Jr, J. H. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, *58*(301), 236-244.

Weidman, S. (2019). *Deep Learning from Scratch: Building with Python from First Principles*. " O'Reilly Media, Inc.".

Whitson, C. H., & Brulé, M. R. (2000). *Phase behavior* (Vol. 20). Henry L. Doherty Memorial Fund of AIME, Society of Petroleum Engineers ….

Wilson, G. (1968). A modified redlich–kwong eos, application to general physical data calculations, paper no. 15c. AIChE 65th National Meeting,

Wu, Y., Kowitz, C., Sun, S., & Salama, A. (2015). Speeding up the flash calculations in two-phase compositional flow simulations–The application of sparse grids. *Journal of Computational Physics*, *285*, 88-99.

Xu, K., & Darve, E. (2020). Physics constrained learning for data-driven inverse modeling from sparse observations. *arXiv preprint arXiv:2002.10521*.

Young, L. C., & Stephenson, R. E. (1983). A generalized compositional approach for reservoir simulation. *Society of Petroleum Engineers Journal*, *23*(05), 727-742.

Yucesan, Y. A., & Viana, F. A. (2019). Wind Turbine Main Bearing Fatigue Life Estimation with Physicsinformed Neural Networks. Annual Conference of the PHM Society,

9. VITA

Thelma Anizia Ihunde received her Bachelor of Science in Petroleum Engineering from the Louisiana State University in 2017, a Master of Business Administration in 2019 and anticipates graduating with a Master of Science in Petroleum Engineering in May 2022.