Anders Bendiksen

Project Thesis
Spring 2021

Creating a machine learning based surrogate model for robust representation of an eNRTL based thermodynamic model of aqueous Pz, AMP and CO_2

Abstract

The purpose of this specialization project is to create a surrogate model of an already existing model that models the vapour-liquid equilibrium of an aqueous system of AMP/Pz/CO_2. When comparing the two models against each other, and experimental data, the results are very promising. The only thing the surrogate model fails to predict with reasonable accuracy is the heat of absorption at higher temperatures, but one should also consider that the base model it had been trained on failed at these predictions too.

Contents

Introduction

Climate change and global warming are among the greatest challenges we as a species, currently have to face. Mitigating and containing the effects of global warming to under 1.5 degrees celsius as outlined in the Paris Agreement is absolutely imperative. CO_2 is the anthropogenic greenhouse gas with the most effect on the gradual heating of the climate,[1] and because of this, the CO_2 emissions need to be cut drastically. There are many promising methods to reduce the CO_2 emissions, the most obvious solution is certainly to cut back consumption and to rely on sustainable sources. This will however take time, and alternative ways to curb the greenhouse emissions must be used in the meantime. CO_2 scrubbing using amines provides a solution that can be conveniently retrofitted[2] to already existing plants, and is among the most mature alternatives for removing CO_2 from flue gas[3].

In order to facilitate easier implementation of such scrubbing, one must have working models of the system. The usual way of doing this is to combine governing thermodynamic relations with empirically fitted parametres. One example of such a model is the eNRTL model for aqueous mono-ethylene-amine (AMP) and piperazine (Pz) elaborated further in Hartono et al 2020[4]. In this specialization project however, statistical modelling will be used to create a surrogate model approximating that model. The main advantage of using this surrogate model is mostly that it is less computationally complex, and thus faster.

1 Theory

1.1 CO2 absorption

The most common way of removing CO_2 is after the combustion has already taken place[3]. The partial pressure of CO_2 is often at it's highest right after combustion, before it has been contaminated with air. This is optimal as CO_2 capture is most energy efficient when the partial pressure is high[5]. CO_2 is absorbed in the aptly named absorber column, usually by aqueous amines. The advantage of aqueous amines, is that the process can be easily reversed, most often by either shifting the temperature, or the pressure. What this means in practice is that the CO_2 absorption itself releases energy, whilst the CO_2 desorption requires energy. The regenerated solution is then returned to the absorption column where the process is repeated. There is however an energy penalty related to regenerating the aqueous solution, usually somewhere in the ballpark of 30% of the, as it currently stands energy originally produced by the power plant[6]. Of course this figure can be reduced by rigorous experimentation and improvement through innovation. Another great advantage that the amine scrubbing method has compared to other methods, is that it can more readily be retrofitted to an already existing plant[2].

In figure 1 a simplified flow diagramme of the absorption process can be seen. At first, the CO_2-rich flue gas is fed into the bottom of the absorber column. Here it meets the lean amine absorbent, usually sprayed from the top of the column to maximize the surface interface area. In the stripper column, the rich amine solution is sprayed from the top of the column, where it meets heated steam from the reboiler, this then reverses the reaction and releases the CO_2. Which is subsequently removed from the top of the column. The condenser makes sure that the aqueous amine solution is returned to the column. In the bottom of the stripping column, the reboiler adds the heat necessary for the reversion reaction to occur. The lean aqueous amine solution is also removed from the bottom of the column, and a heat exchanger is used to minimize the heat wastage by heating the rich amine solution coming from the absorber column. The lean amine is then pumped back into the absorption column.

There are two main mechanisms for aqueous absorption of CO_2, physical and chemical. Physical absorption is when the CO_2 is dissolved whithin the solvent itself, and chemical absorption is when chemical agents are added to the solvent

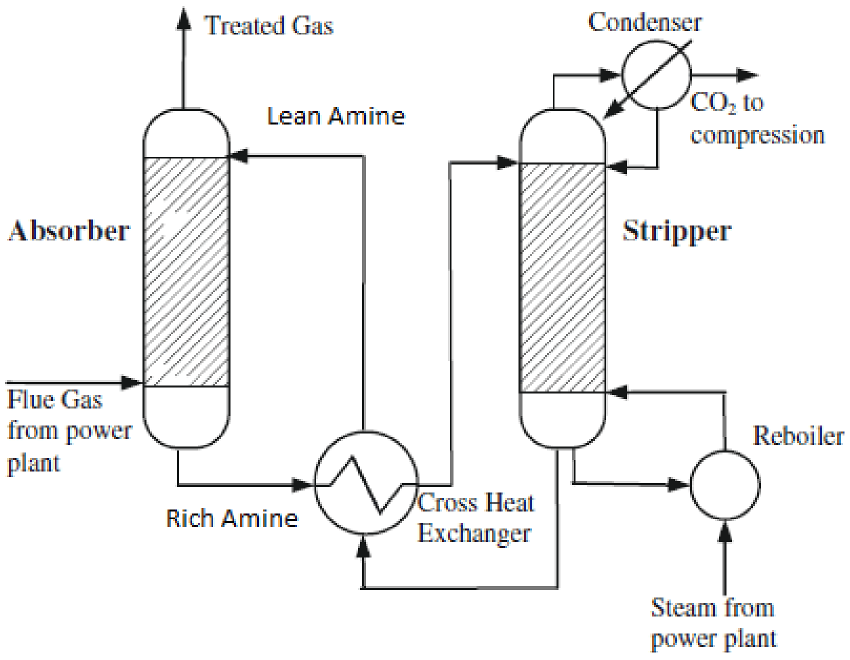


Figure 1: A process flow diagramme of the CO_2 absorption process. From Li et al. 2017[7]

that react with the absorbed CO_2 and thus removing it from circulation and further facilitating physical absorption of CO_2. For the purposes of this work, the system uses a combination of both physical and chemical absorption.

1.2 Introduction to statistical modelling

Neural networks were developed independently by statisticians and data scientists from a technique known as project pursuit regression[8]. Project pursuit regression works by minimizing the error of the expression given in equation 1.

$$f(X) = \Sigma_{m=1}^{M} g_m(\omega_m^T X) + \varepsilon \tag{1}$$

In order to train the above model, we seek to minimize the error function every time we adjust the parametres. In most cases, this error function is the mean square error, given in equation 2. In the expression above, the $g_m()$ is a ridge function working on the $\omega_m^T X$ component, whilst the $\omega_m^T X$ itself is the projection of X onto the vector ω_m. The training process then seeks to find a vector ω_m that best matches the model and minimizes the error ε, hence the name; projection pursuit.

$$MSE = \Sigma_{i=1}^{K} (y_i - \hat{y}_i)^2 \tag{2}$$

The projection pursuit model is then trained by minimizing the following cost function.

$$\Sigma_{i=1}^{N} [y_i - \Sigma_{m=1}^{M} g_m(\omega_m^T x_i)]^2 \tag{3}$$

While project pursuit regression is not directly relevant to what this work is about, it serves as an illustrating example on how to better understand neural networks, how they work and how they came to be. Further information about projection pursuit regression can be found in Hastie & Tibsharanie[8].

1.3 Neural Networks

Neural networks are often discussed as though they are mysterious entities capable of solving humanity's most difficult problems. They are however, nothing more than non-linear statistical models. They can be used for both regression and

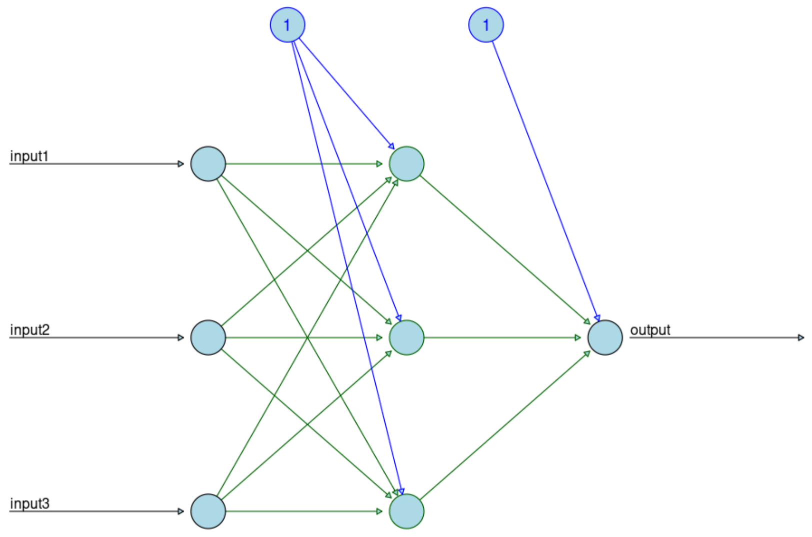


Figure 2: A simple neural network with three inputs, one response and a single hidden layer with three perceptrons. The nodes marked ”1” are bias nodes

classification, and as such are supremely versatile, capable of being utilized in a plethora of different situations. Neural networks derive their name from the fact that they are connected in a similar manner to neuron cells in a human brain[8].

In figure 2, a simple neural network, with a single hidden layer, three inputs and one response can be seen. As can be induced from a quick glance, the hidden layer contains three perceptrons. The hidden layer is the middlemost layer, between the input an the output layers. Perceptrons are the blue circles in the hidden layer and the output layer. A perceptron applies an activation function to the inputs, adds them together and then passes them on to the next layer. The input nodes are not classified as perceptrons, since they only pass on the unadulterated inputs to the next layer. The bias nodes marked ”1”, which incidentally is also their input, serve as an independent input to each variable. Eensuring that all of the perceptrons have at least some input and preventing ”dead perceptrons”. The weights of the bias nodes are optimized in the same way as the other weights of the network. The role of the bias is equivalent to that of the intercept in a linear expression, and must not be confused with the model bias that is discussed in section 1.3.2.

The process done by a single perceptron is given in equation 4.

$$Output = Activation\,Function(\Sigma(Weights \cdot inputs)) \tag{4}$$

The neural network is a so called black box model, and as such there is no way to deduce the physical meaning of the weights[9]. This means that while a neural network may be a good approximation, one must always be careful to ensure that the model returns meaningful responses.

1.3.1 Activation functions

Activation functions are usually expressed as σ, and they can be essentially anything the creator of the neural network wants them to be, of course with a few limitations.

$$\sigma(x) = f(x) \tag{5}$$

In a very simple neural net, appropriate for approximating linear relations, one would use a linear function of the form $f(x) = ax + b$ as the perceptron activation. The role of this activation function is to transform the perceptron inputs into the output. However, using neural networks to approximate or predict linear relations is utterly uninteresting, as the resulting model would be nothing more than a linear regression model. It is then imperative that one uses a non-linear activation function. In short, the activation function simply takes the combined sum of the inputs and the bias, and applies an activation function to it.

$f(x)$ can be basically whatever the creator of the network wants. One of the most common is the sigmoid function, as expressed below in equation 6.

$$f(x) = \frac{1}{1+e^{-x}} \tag{6}$$

As we can see here, the Sigmoid activation function is a lot more complex than $f(x) = ax + b$, and as such, it allows the network to predict non-linear relations. Another common activation function is the hyperbolic tangent function.

One of the most common activation functions for regression[10] is the rectified linear unit, or RelU function. This function can be seen in equation 7, and a plot in figure 3. The Rectified Linear Unit's main advantage is that it does not

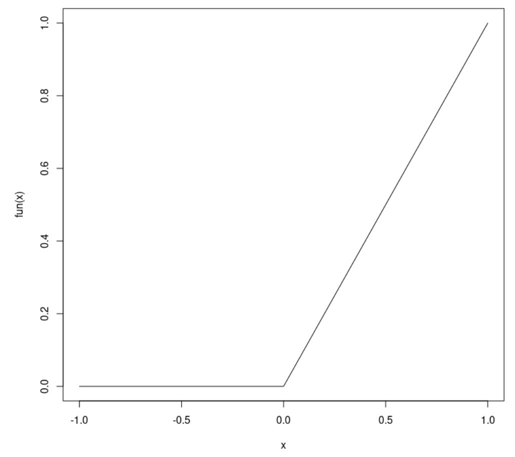


Figure 3: A plot of the rectified linear unit

activate all the perceptrons at once, and thus is capable of approximating non-linear trends. It is also very simple, and in this way prevents the model from becoming too complex and ensuring a quick training period. A slightly modified version of the Relu function, called leaky ReLU, has a very slight slope instead of the zero value. This is because the Relu might get ”stuck” in a local minima sometimes, and the slope is useful for giving the model at least some input. This leaky ReLU function is given in equation 8.

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{7}$$

$$f(x) = \begin{cases} 0.00000001x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{8}$$

1.3.2 Bias-variance trade-off

When training models, one must take care not to overfit the model to the training data. Overfitting will invariably result in the model performing very well on the training dataset, but failing to perform adequately on new validation data.

In the illustration in figure 4, the green line is overfitted to the training data. One can observe that the black line, while performing a lot worse on the training

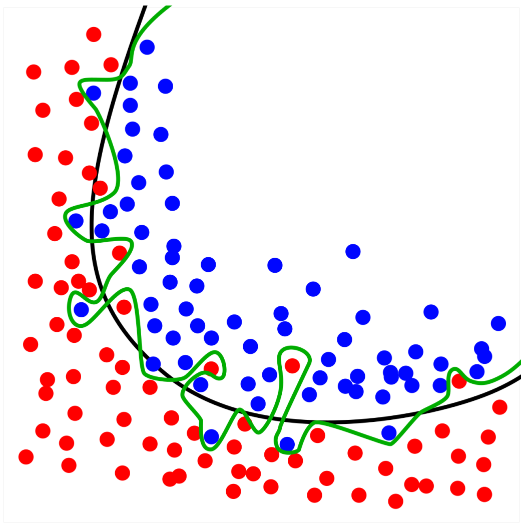


Figure 4: An illustration of overfitting. Picture by Ignacio Icke, distributed under a CC BY-SA 4.0 license.

dataset, is a lot better overall, and probably would do a better job at classifying new data. Overfitting was a constant problem plagueing early neural networks, one of the first fixes was stopping the propagation early[8] and simply hoping that this was good enough. Usually a validation dataset was used to find the optimal stopping point. After a while however, more sophisticated methods were developed.

There are two common ways to avoid overfitting a neural network, these are regularization and *dropout*. *Dropout* will ignore a certain percentage of the perceptrons, and simply set their output to zero[11]. This is a surprisingly effective way of preventing the network of becoming too dependent on a few single perceptron combinations. The other way to avoid overfitting is regularization. This method works by adding a penalty term to the residual function of the neural network. This is expressed in equation 9, where $R(\theta)$ is the original residual function, and $\lambda J(\theta)$ is the penalty term. λ is a tuning parameter and $J(\theta)$ is either the expression from equations 11 or 10.

$$R(\theta) + \lambda J(\theta) \tag{9}$$

$$J(\theta) = \Sigma_i |W|_i \tag{10}$$

$$J(\theta) = \Sigma_i W_i^2 \tag{11}$$

There are multiple ways to regularize, the two methods that are relevant to us are the L1 and L2 regularization methods. They are quite similar, with only a slight difference. L1 regularization adds a single W-term to the cost function, while L2 regularization adds a W^2 term. They differ in that L1 will estimate around the median of the data, while L2 will estimate around the mean of the data[11]. This can have profound effects depending on how the data is distributed.

1.3.3 Training neural networks

When training a neural network, simple backpropagation is the most common method. Backpropagation, which is shorthand for backwards error propagation, works by computing the gradient of a loss function. And then gradually updating the weights to minimize the loss function along the gradient. The updating of the weights is done every time new data is propagated through the neural network. The cost function is often the mean square error, given in equation 12, or the mean absolute percentage error, as given in equation 13.

$$MSE = \frac{1}{N} \Sigma_{i=1}^{K} (y_i - \hat{y}_i)^2 \tag{12}$$

$$MAPE = \frac{1}{N} \Sigma_{i=1}^{K} \frac{|(y_i - \hat{y}_i)|}{y_i} \tag{13}$$

In order to find the gradient of the loss function, an optimization algorithm must be chosen. The most natural choice is the gradient descent. Gradient descent finds the gradient by following the gradient of the cost function down the steepest slope, and in this way finding the minimum of the cost function. In order to limit the rate of change of the weights, one can implement a constraint called the learning rate. A too high learning rate will make the weights fluctuate and produce an unstable model. While a too low learning rate will maybe get stuck in a local minimum. Gradient descent[8] can be expressed as done in equation 14.

$$b = a - \gamma \nabla f(a) \tag{14}$$

Here, b represents the next position, a represents the current position, γ is the learning rate, whilst the $\nabla f(a)$ is the direction of descent.

Stochastic Gradient Descent, or SGD builds on the main idea of ordinary gradient descent, but also adds an element of randomness, which is what ”stochastic” means. Instead of computing every data point individually as in ordinary gradient descent, data points are selected at random. This is of course not as accurate as gradient descent, but it is a great deal faster. We will however not really use SGD, and the only reason for writing about it is because is serves as a great way of illustrating how the training of a Neural Network works. The normal way of choosing a optimizer is to choose the latest and greatest, and double check it against other tried and true optimizers to see if it truly is the best for that particular model. Cross-referencing against other methods is important since statistical modelling is no exact science, and when in doubt, it is best to have data to back up the assumptions made for that particular model.

ADAM, or adaptive momentum, builds upon the SGD optimizer, but it also incorporates an adaptive learning rate and momentum. This optimization method is new, being only introduced in 2015[12]. The adaptive learning rate works by adjusting the learning rate as the slope changes. Whilst the momentum works by adding a ”temporal element” to the SGD algorithm that ”hastens” the convergence along the steepest descent. A common analogy is that of a boulder rolling down the path of least resistance. More information about how ADAM works can be found in Kingma & Ba[12].

1.4 Enthalpy of absorption

The values estimated from the surrogate model can be used to calculate the heat of absorption. By modifying the van’t Hoff equation we get the expression given in equation 15. For further information about how it is derived read Svendsen et al, 2011[13].

$$\frac{\partial ln(P_{CO_2})}{\partial(1/T)} = -\frac{\Delta H_{CO_2}}{R} \tag{15}$$

This equation can then be used to find the heat of absorption as a function of partial pressure of CO_2 and the temperature. The heat of absorption constitutes one of the largest operating expenditures of a CO_2 scrubbing plant, and as such, the heat of absorption is quite important.

2 Model Building

When building a neural network based model, there are a great many considerations and choices to make. In this section, these considerations will be further explained and the necessary context added.

The creation of the surrogate model consists of many steps, and to present exactly what goes into this model in a comprehensive and understandable fashion, a short summary can be found just below.

1. **Selection of base model:** One must find a working thermodynamic model with which to train and compare the surrogate model
2. **Degree of freedom analysis:** Certain thermodynamic constraints must be satisfied in order for the system to converge properly
3. **Parametres of the neural network:** The neural network naturally contains many parametres, in this section these will be explained further
4. **Generating and preprocessing the training and validation datasets:** The datasets used for the model must be created, and then processed so that the model will accept them
5. **Training the model:** The neural network must be trained on the training dataset in order for it to accurately predict the values of the base model
6. **Validating the model:** In order to ascertain that the model actually is accurate the model must be validated against an independent dataset

Variable	Lower range	upper range
Loading [α]	10^{-4}	1.1
Piperazine [wt%]	10^{-4}	15
AMP [wt%]	10^{-4}	35
Temperature [Celsius]	20	120

Table 1: The limits of the training data set

2.1 Base model

In order to generate data that the neural network can be trained on, a base model is needed. This base model needs to accurately predict the empirical values of the system and model the AMP-Pz-CO_2-H_2O vapour/liquid equilibrium (VLE). This base model would then be constructed in the same manner as the surrogate neural network, requiring the same independent variables and returning the same dependent variables.

The eNRTL model is a framework for calculating the activity coefficients of an aqueous system, the specific method is outlined in Chen et al.[14] In short, it works by correlating the deviation from ideality, i.e. the activity coefficients, to the excess Gibb's energy in a multicomponent electrolytic system. This model can then be used in conjunction with the Peng-Robinson equation of state to model both the two phases of the system. For more information about how to implement this specific model to an aqueous system with AMP and Pz, see Hartono et al (2020).

This base model has certain limitations when it comes to the ranges of the independent variables, since the surrogate model has not been trained on data outside of these ranges, any input not within the table below would produce wildly inaccurate results, and should be taken with a large fist of salt. Common sense then dictates that the model only be used within the bounds specified by the eNRTL model. These limits are given in table 1.

2.2 Degree of freedom analysis

Gibb's phase rule for reactive systems is given as follows

$$F = 2 + C - P - r - s \tag{16}$$

In equation 16, the F is the degrees of freedom, the C is the minimum number of components, the P is the number of phases, the r is the number of independent reactions and s is the number of summation constraints. The training of neural network models is a finicky process. If not sufficient care is taken, the model can end up converging towards a completely different point than one initially wanted (Carranza-Abaid et al. 2020). In order to rectify this problem, one should specify the system fully according to Gibb's phase rule for reactive systems. This allows us to find the correct number of independent variables.

There are 15 species in total in the system, as specified in table 2 and 3. 10 independent reactions, given in equations 17 to 26. Two phases and a single constraint. The constraint is that the ionic charges in the system at thermodynamic equilibrium, must add up to zero. A cursory calculation then gives us that the number of independent variables must be 4. The independent variables must also be chosen with care, and the most natural choice falls upon using the loading of the CO_2, the temperature and the amount of the two amines in the mixture. The amount will in this case be expressed as a weight percentage.

$$2H_2O <=> H_3O^+ + OH^- \tag{17}$$

$$CO_2 + 2H_2O <=> H_3O^+ + HCO_3^- \tag{18}$$

$$HCO_3^- + H_2O <=> H_3O^+ + CO_3^{2-} \tag{19}$$

$$AMPH^+ + H_2O <=> AMP + H_3O^+ \tag{20}$$

$$H_2O + CO_2 + AMP <=> AMPCO_2^- + H_3O^+ \tag{21}$$

$$PzH^+ + H_2O <=> Pz + H_3O^+ \tag{22}$$

$$H_2O + PZH_2^{2+} <=> PzH^+ + H_3O^+ \tag{23}$$

$$H_2O + CO_2 + Pz <=> PzCO_2^- + H_3O+ \tag{24}$$

$$Pz(COO^-)_2 + H_3O^+ <=> PzCOO^- + CO_2 + H_2O \tag{25}$$

$$PzH^+COO^- + H_2O <=> PzCOO^- + H_3O^+ \tag{26}$$

Combining equations 20 and 24 and equations 20 and 25 gives us the following:

$$CO_2 + AMP + Pz <=> PzCO_2^- + AMPH^+ \tag{27}$$

$$CO_2 + AMP + PzCO_2^- <=> Pz(CO_2^-)_2 + AMPH^+ \tag{28}$$

The reactions given in equation 27 and 28 are the combined reactions for CO_2 absorption in the system.

As shown in tables 2 andn 3, we need 15 species to fully characterize our system. All of them are present in the aqueous phase, whilst only the first four are presumed present in the gaseous phase.

2.3 Model building

The interface used to build the model was the R version of Tensorflow. The Nvidia Computational Unified Direct Architecture (CUDA) was used, or more specifically, the CUDA Direct Neural Network Library (CuDNN) was used to perform the calculations in parallell. This drastically cuts down the time used for the calculations and training the neural network. This speed advantage is the main reason for choosing this framework. It is also infinitely flexible since it can be combined with every R and Python library. The model itself was trained on an Nvidia RTX 2070 GPU using the latest drivers and CUDA 10.1.

Species present in the aqueous phase
H_2O
CO_2
$Piperazine$
AMP
H_3O^+
PzH^+
$AMPH^+$
PzH_2^{2+}
OH^-
HCO_3^-
CO_3^{2-}
$PzCO_2^-$
$Pz(CO_2^-)_2$
$AMPCO_2^-$
$PzH^+CO_2^-$

Table 2: The species present in the aqueous phase

Species present in the gaseous phase
H_2O
CO_2
$Piperazine$
AMP

Table 3: The species present in the gaseous phase

2.4 Computational considerations

For a neural network with N observations, p predictors, M hidden units and L training epochs, the model the number of operations given in equation 29.

$$N_{operations} = \Omega(N \cdot p \cdot M \cdot L) \tag{29}$$

One can therefore see that the model has the potential to become rather large. Care must be taken so that the model. However the implementation of parallel computing within the CuDNN framework does negate some of the incurred time penalty of a more complex model.

2.5 Model building and parametre adjustment

Statistical modelling is, by it's very nature a highly empirical field. One cannot know for certain what parametres or methods will work best without trying and evaluating every single one of them. Of course an exhaustive study of every statistical model known to mankind would be out of the scope of this specialization project. And as such, certain time saving measures, such as automatically assuming that a shallow neural network is the best method for approximating[15] the eNRTL model have already been made. Other choices were also made in the interest of keeping the model as computationally simple as possible, whilst still retaining as much accuracy as possible. When these choices have been made, they will be justified and discussed in the results section of this thesis.

The best way to find out what parametres fit the model best, is to try everything. And then cross validate against an independent validation data-set. As for the assumption that a shallow neural network is sufficient for approximating this specific model.

2.5.1 Choice of loss function

When choosing the loss function of a neural network model, one must consider the peculiarities of the dataset. In this case, since the outputs of the network vary wildly in magnitude (most of them being molar fractions between 0 and 1, and one of them being the pressure of the system). Choosing the mean square error would mean that the model would be biased towards predicting the response of the pressure more accurately than the other responses. Of course, whilst normalizing the data does prevent some of this effect, choosing the mean average percentage error, as given in equation 12, can also effectively counteract this bias, since every error will be a relative fraction of the variable error instead of the absolute error of the variable. This will spread the effects of the cost function minimalizing more equitably between the different response variables.

2.6 Creating and normalizing the data sets used

In order to create the data set, the eNRTL model mentioned above was used. As we have used CuDNN, optimizing the training of the neural network for speed is not really a priority. This is because the model will be plenty fast anyway, and compromising the accuracy of the model for a few seconds less training time is not seen as a necessary sacrifice. Therefore 10000 observations, randomly distributed along the criteria outlined in table 1, were used to train the model. 8000 of them for the training, and 2000 for validating. These are also randomly distributed into the two data-sets.

When training a neural network, it is very important that the input and output are the the same domains. Widely varying inputs and outputs would need a more complex and difficult to train network. For instance, the molar fractions are all between 0 and 1, whilst the pressure is between 0 and several hundred pascal. By normalizing the values, so that they are circa in the vicinity of each other, the network incurs less of a computational cost, and is usually more accurate. There are multiple ways to process the the independent variables before they are fed into the network.

$$X_{normalized} = \frac{X - X_{max}}{X_{max} - X_{min}} \tag{30}$$

Performing this normalization should transform our independent variables to lie between 0 and 1. For this specific model, we have used a different type of normalization. The mean and standard deviation is found for the different independent variables, and the variables are then normalized along a standard normal distribution with the standard deviation of the original distribution.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{31}$$

In equation 31[16], x is the value to be normalized, μ is the average of the variable and σ is the deviation of the variable.

In order to normalize the output of the neural network the natural logarithm of every response variable is used instead of the actual value, this is done to counter the effects of the varying range of the variables. There is no special reason that the normalization method used was chosen over any other method. They worked well in practice and were superior to the other methods of normalizing. The specific results and justification will be presented later on in the thesis.

2.7 Training the model

In this step, the independent variables are matched to the dependent variables. And the weights of the neural network updated in the manner that minimizes the cost function. The model itself was implemented in tensorflow, and the code can be found in the appendix.

2.7.1 Epochs and batch size

When training a neural network, one must also specify the number of iterations, or epochs, the algorithm should run. In the figure below, the error is plotted against the number of epochs. One can of course create an algorithm that shuts down as soon as an acceptable training error has been achieved, but for this model. This approach was deemed too complex without really giving any tangible benefit, and a fixed epoch number of 1000 was decided upon. This may or may not have been overkill, but since the model itself was so fast it was not seen as necessary. Especially as the cost function was still being minimized until around 800-900 epochs.

A batch is merely a group of observations used together when training an epoch. Having a small batch size is easier computationally, but can introduce erratic and unpredictatable patterns into the model. Subsequently, a batch size of 120, an intermediate size was chosen, mainly as a compromise.

2.8 Validating the model

When validating the model, an independent dataset was used. As the normalization presumably makes the model susceptible to differently distributed datasets, a worst case scenario was simulated by constraining the validation dataset around a

Independent Variable	Constraint
Loading	$10^{-5} \rightarrow 1.1$
Piperazine [weight %]	9.5
AMP [weight %]	15.5
Temperature [Celsius]	100

Table 4: The limits of the validating data set

few parametres. And as seen later on in the results section, this did indeed produce a higher error than a perfectly distributed set. The constraints used are specified in table 4.

3 Results and discussion

In this section, the results will be presented and discussed.

3.1 Model parametres

As discussed earlier in the section about model building, there are a great many parametres to determine in order to find an appropriate surrogate model. When choosing model parametres, the best way is usual by trial and error. For instance, instead of using abstract theory to determine the optimal amount of perceptrons, one could just try all the possible combinations of parametres and use the best one. This might sound tedious, we cannot ascribe a physical meaning to the weights of the neural network, there is little else to do.

3.1.1 Number of perceptrons

To find the data in figure 5. The model was run 10 times for each number of perceptrons, whilst the number of perceptrons was changed in increments of 5. While there is an argument to be made that starting at 80 perceptron is perhaps a smidgin too high, the model itself was so fast that reducing the complexity by keeping the number of perceptrons low, was not seen as necessary. As one can see from the plot, the training error was consistently low, but the validation error seems to indicate that a single hidden layer with 110 perceptrons was the optimal choice with regards to error. As you can see, at 110 perceptrons, the validation error has a local minimum. And even though the validation error is the lowest at 135 perceptrons, and the training error the lowest at 130. It is often not worth further complicating the model for a miniscule amount of accuracy. Therefore 110 perceptrons is a good compromise between accuracy and complexity. The error is of course not the only consideration one should have in mind when constructing a model, but the model complexity will be discussed later.

While there is no special consideration that made a single player the obvious choice, there is often little reason to further complicate the neural network if it is accurate enough. And as will be shown further on in this report, there is little need for a more complicated model.

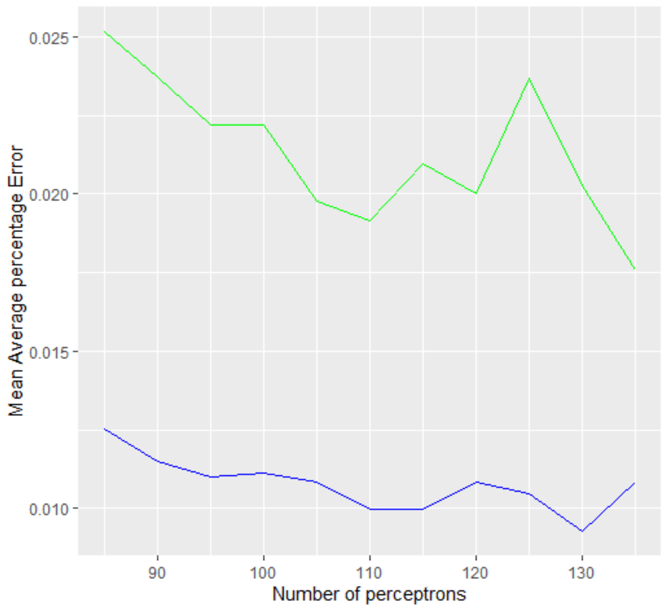


Figure 5: The training and validation error plotted against the number of perceptrons

3.1.2 Regularization method and learning rate

Dropout was the first method employed to avoid overfitting the network. This however proved to be rather unsatisfactory, as it introduced too much error and unpredictability, often in the ball park of 8%-9%. The use of regularization on the other hand proved to be entirely satisfactory for the purposes of this model.

As can be seen in figure 6, almost all the possible alternatives are usable. The L1 regularization method with a learning rate of 0.005 was chosen, since it performed ever so slightly better than the others.

3.1.3 Activation function

When choosing the activation function, the most obvious choice was the REctified Linear Unit, or RelU. It is further explained in equation 7. The other functions considered were the sigmoid function, the hyperbolic tangent, and lastly a modified Relu called the Leaky Relu function, given in equation 8. Here only 5 iterations for every activation function was used, because the large discrepancy in errors showed that only the Relu was an acceptable alternative. As can be seen in

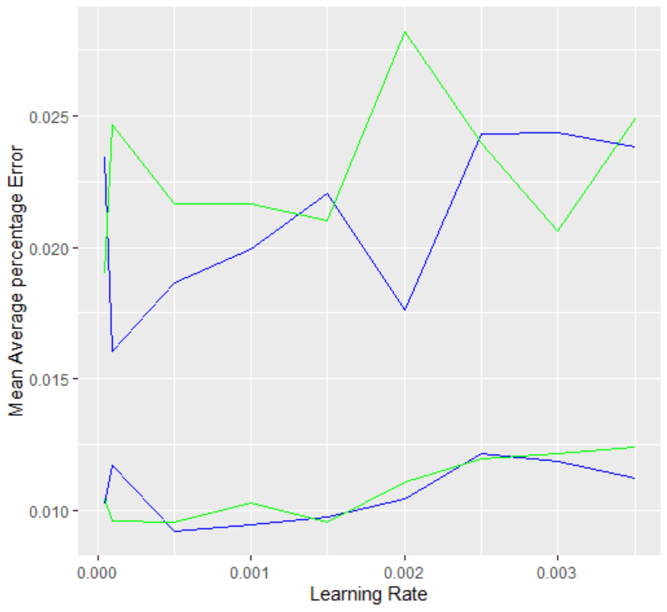


Figure 6: The L1 and L2 error plotted against the number of perceptron. The two lowermost lines are the training error, and the two uppermost the validation error

figure 2, an activation function is also applied to the output node. For this model, these specific activation functions were set to be linear, since the non-linear approximations take place in the hidden layer and there is no need for further complexity.

The table above contains the average of the validation error for the four different activation functions. Of the four, the Relu function is the one that showed the most promise when comparing errors, and as such was the one used.

Activation Function	MAPE Validation Error
ReLU	0.020889274
Leaky ReLU	0.039646033
Sigmoid	0.142391833
Tanh	0.100044418

Table 5: The MAPE errors of the different activation functions

Variable	MAPE training Error	MAPE validation error
χ_{H_2O}	0.00028	0.00178
χ_{CO_2}	0.00928	0.01597
$\chi_{Piperazine}$	0.00757	0.00845
χ_{AMP}	0.01241	0.01045
$\chi_{H_3O^+}$	0.01366	0.02250
χ_{PzH^+}	0.00405	0.01456
χ_{AMPH^+}	0.01567	0.01168
$\chi_{PzH_2^{2+}}$	0.01106	0.01232
χ_{OH^-}	0.01306	0.05083
$\chi_{HCO_3^-}$	0.00219	0.01328
$\chi_{CO_3^{2-}}$	0.00047	0.00153
$\chi_{PzCO_2^-}$	0.00189	0.01376
$\chi_{Pz(CO_2^-)_2}$	0.00989	0.00609
$\chi_{AMPCO_2^-}$	0.00729	0.02212
$\chi_{PzH^+CO_2^-}$	0.01160	0.01142
y_{H_2O}	0.00254	0.09875
y_{CO_2}	0.00348	0.01494
y_{Pz}	0.02548	0.00799
y_{AMP}	0.00843	0.02876
Pressure	0.00151	0.04056
Total for Model	0.00761	0.02053

Table 6: The different errors of the model

3.1.4 Training and validation error

As can be seen in both table 6, and in multiple other places in this project, there are a great many dependent variables. In the interest of presenting a coherent and understandable thesis, some of these will be therefore be disregarded in the name of brevity. The response variables that are relevant, will on the other hand be presented and discussed in exhaustive detail. The most interesting response variables for modelling the absorption itself are the molar fractions of CO_2 in the gaseous phase and the pressure of the system. These can then be used to calculate the heat of absorption and the partial pressure of CO_2. The amount of CO_2 absorbed is already defined by the loading, and as such, already accounted for. Another reason for the choice of these specific variables is the relative abundance of experimental data available.

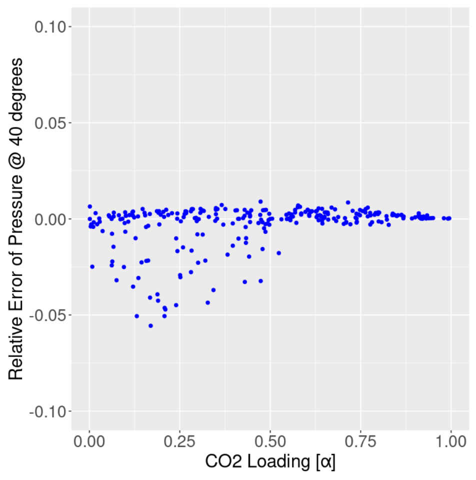
As one can see in table 6, the training error of the model is just 0.76%, a very acceptable number. The validation error, is also at a very acceptable 2%. This value should not be a cause for concern as it represents a worst case scenario. Some of the values are of course better than others, but comparing them serves first and foremost as a way to judge the overall error of the model and establish a ballpark estimate of how large the error truly is. For a better comparison of the error, one should plot it against the independent variables.

3.2 Residual and relative Error

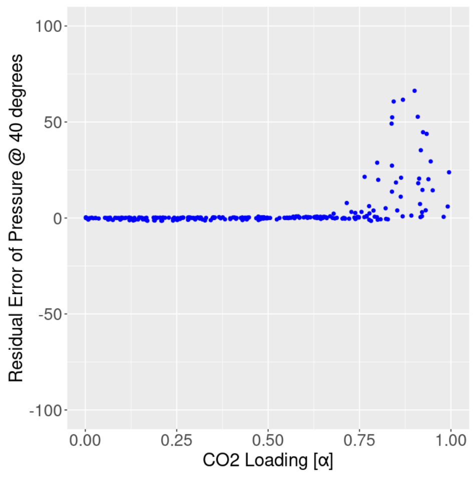
To calculate the relative error, a modified version of the MAPE error given in equation 13 was used. The difference being that in this specific case, we did not use the absolute value. For the residual error the following equation was used: $y_{base} - y_{surrogate}$. In this manner both the actual numerical value of the deviation, and the deviation as a fraction of the total value could be compared. This is important to establish context regarding the usability and accuracy of the surrogate model.

In figures 7 to figure 12, the relative and residual errors for both the total pressure of the system, and the gaseous molar fraction of CO_2 are presented. For these plots, only the temperature is constrained, while all other independent variables are kept as they are given in table 1. A general trend that can the observed in the errors for the total pressure is that at higher temperatures, the model is less accurate. It is worth noting however that, for 60 and 40 degrees the model does perform admirably well.

The relative and residual errors for the pressure do tell different stories though. Figure 7 and figure 8 tells us that at low values for the loading, the relative error is high, whilst at high values the absolute, or residual error is high. This is not unexpected, since lower absolute values of pressure would be more sensitive to a small error. And the residual error at higher loadings, while high, are rather small when compared to the actual value of the pressure. This means that whilst not perfect, the model does behave predictably when given intermediate to low temperature inputs.



(a) Relative error



(b) Residual error

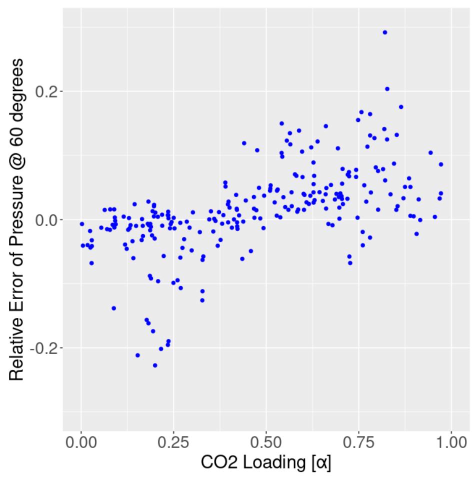
Figure 7: The errors of the total pressure plotted against the loading @ 40 degrees

At higher temperatures the model gives a larger error, most of the observations are however within 15%, with the majority of those again being within 10%. This is not a perfect result, but it is also important to remember that the surrogate model was developed as a faster alternative to the base model, which it certainly excels at. So an error mostly within 15% is perhaps not too bad after all.

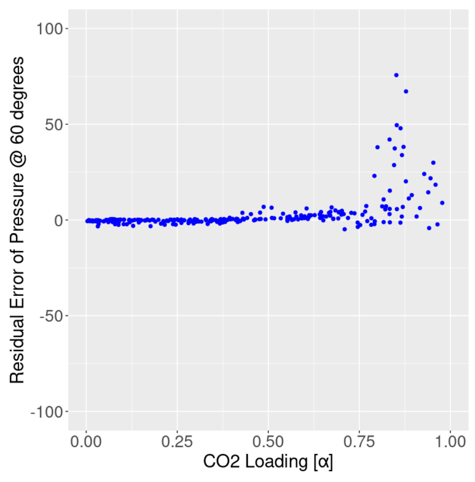
3.3 Partial pressure of CO2 plotted against the loading

In order to compare accuracy of the model itself, it was plotted against experimental data at certain predefined temperatures and concentrations. These results are presented in figures 13 and 14. These expermimental results were taken from Dash et al 2011[17] and Hartono et al 2021. They are taken at a composition of 12.9 w% piperazine and 26.6 wt% AMP for the data in figure 13, and a composition of 8 wt% piperazine and 32 wt% AMP for the data in figure 14.

Here one can see that the model generally does a good job of approximating the eNRTL model. When it does diverge from the model, it's a toss-up whether it improves the model or not. This is as expected, as the neural network showing bias would be a cause for concern. One can also observe the model exhibiting strange

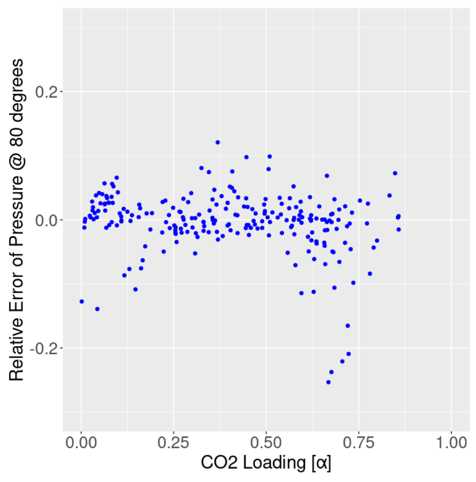


(a) Relative error

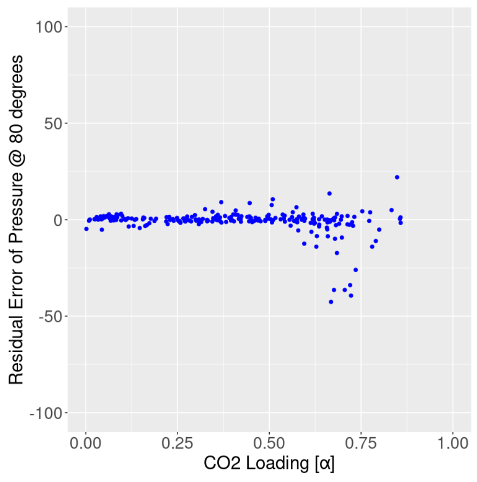


(b) Residual error

Figure 8: The errors of the total pressure plotted against the loading @ 60 degrees

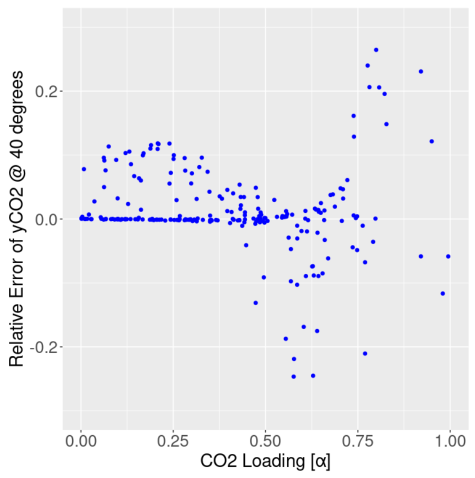


(a) Relative error

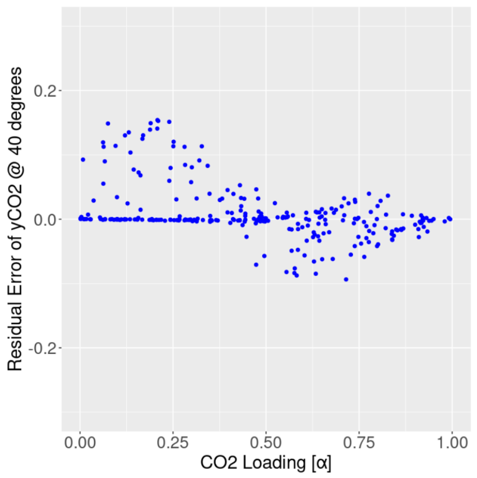


(b) Residual error

Figure 9: The errors of the total pressure plotted against the loading @ 80 degrees

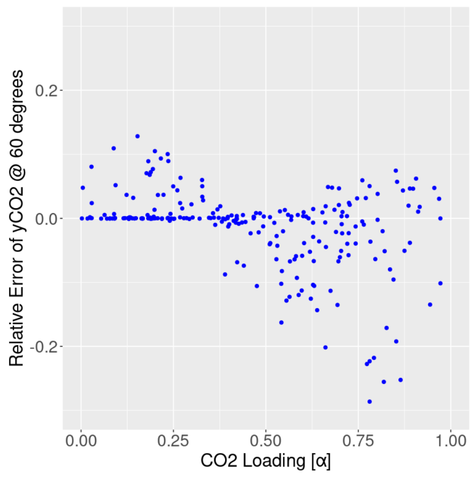


(a) Relative error

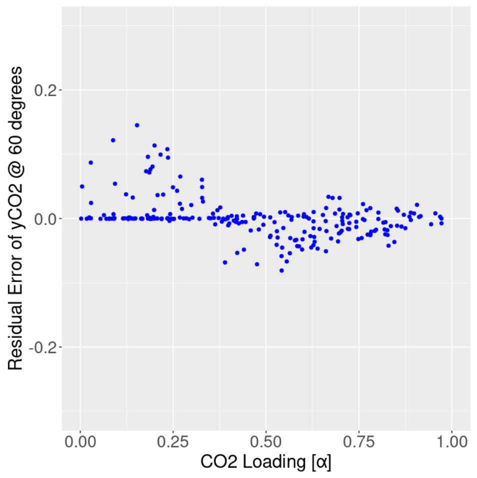


(b) Residual error

Figure 10: The errors of the gaseous molar fraction of CO2 plotted against the loading @ 40 degrees

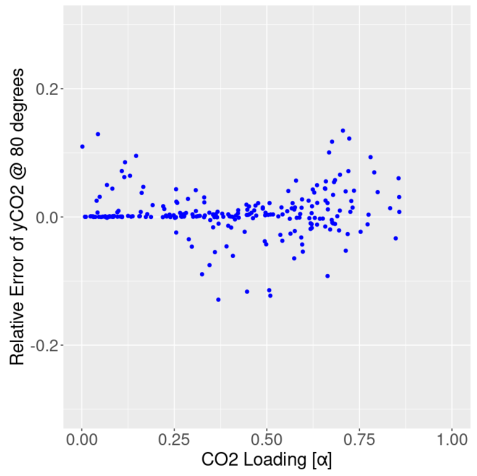


(a) Relative error

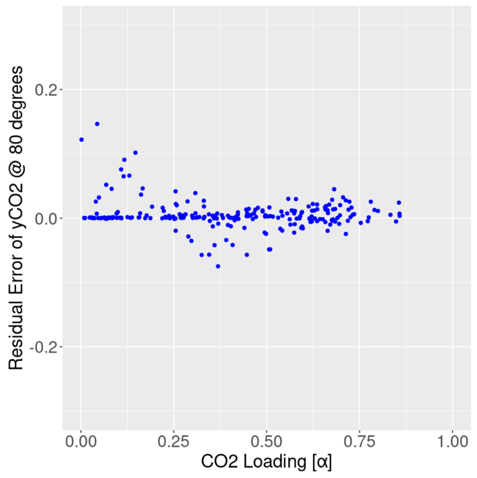


(b) Residual error

Figure 11: The errors of the gaseous molar fraction of CO2 plotted against the loading @ 60 degrees



(a) Relative error



(b) Residual error

Figure 12: The errors of the gaseous molar fraction of CO2 plotted against the loading @ 80 degrees

Temperature	MAPE surrogate model	MAPE base model
40 degrees celcius	0.09318	0.07460
60 degrees celcius	0.13049	0.76234
80 degrees celcius	0.16919	0.10889

Table 7: The deviance from the experimental values at different temperatures

behaviour that takes the form of ”jagged” breaks, This behaviour is especially prevalent in figure 14a and can be seen towards the lower end of the loading.

The only reason that the plots in figure13 and figure 14 are presented differently is that the values in figure 14 were so close together they could not be presented in the same way without causing great confusion. Especially the experimental data in figure 14a was encroaching upon the other plots and making it very hard to read.

3.3.1 Deviation from experimental data

Since it may be difficult to read directly from the plots in this section, the deviation of the two models from the experimental values is expressed in table 7.

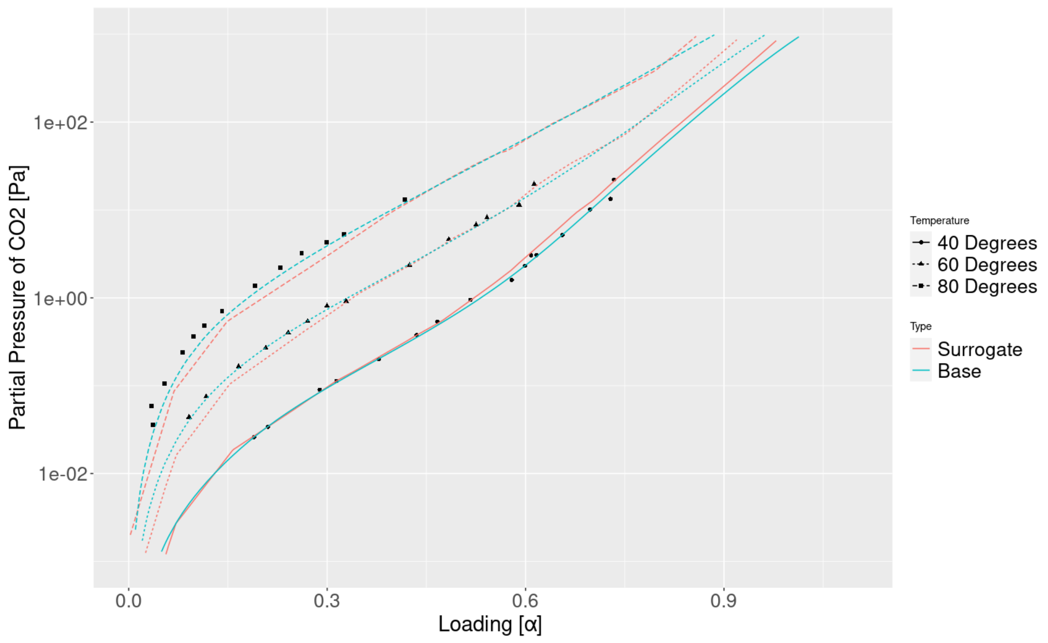
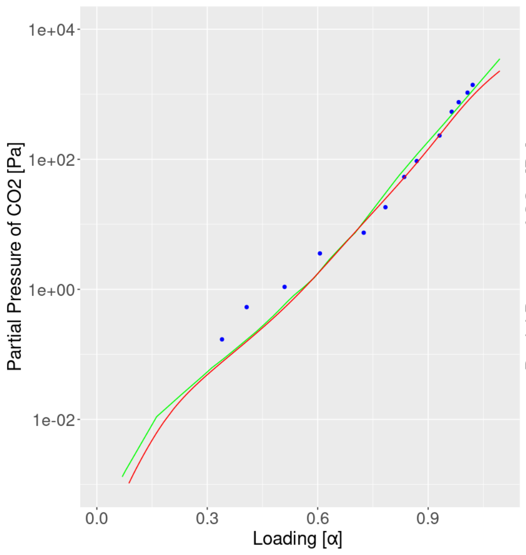
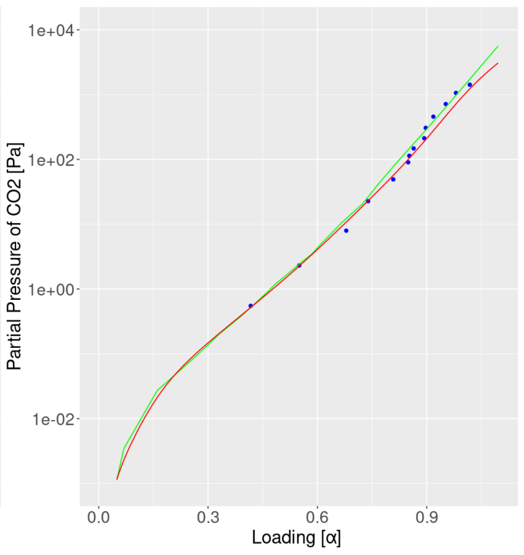


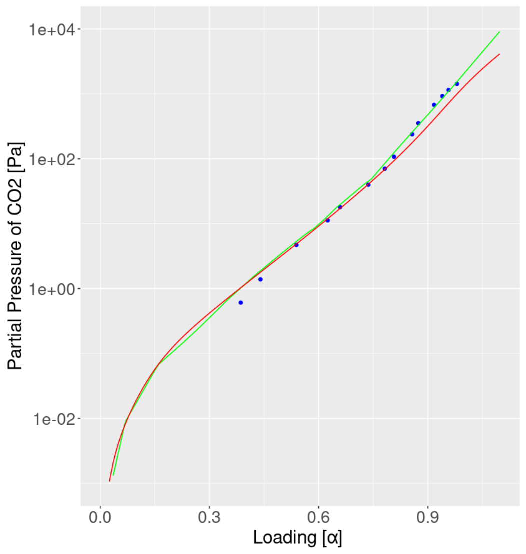
Figure 13: The partial pressure of CO2 plotted against the loading. 12.9 wt% Piperazine and 26.6 wt% AMP. The points are the experimental data while the lines are the predicted data.



(a) 30 degrees celsius



(b) 40 degrees celsius



(c) 50 degrees celsius

Figure 14: The partial pressure of CO2 plotted against the loading at different temperatures. Experimental data, surrogate model and base model are all represented. 8 wt% Piperazine and 32 wt% AMP.

3.4 Heat of absorption

As discussed earlier, the heat of absorption is an integral part of modelling CO_2 absorption. The results, calculated from equation 15 are presented in figures 15 to 17. The experimental data are taken from Xie et al. 2011[18] and the composition is 4.5 w% piperazine, and 27 w% AMP. The temperatures are respectively 40, 60 and 80 degrees celsius.

The model does actually predict the correct values of the heat of absorption at intermediate to low temperatures, and at loading values in the range from 0.2 to 0.8. Luckily for us, extreme values for the loading are not very common because of the extra energy required. The inaccuracy at higher temperatures does however make it more difficult to model the desorption process, as this is naturally done at a higher temperature than the absorption. As can be seen in figure 17, the accuracy of the model for the heat of absorption at 80 degrees celsius is so bad as to render the entire model unusable for that temperature interval. There are probably multiple reasons for this. The most probable reason is that the Gibbs-Helmholtz equation (equation 15) used to calculate the heat of absorption includes some simplifications that are not valid at higher temperatures for this specific mixture.

This seems to mean that while predicting the partial pressure of CO_2 with good accuracy, the eNRTL model itself does not predict the heat of absorption well enough. Now of course, it was never meant to either. The strange thing however, is that the neural network does more often than not predict the heat of absorption with a greater accuracy than the eNRTL model. This almost certainly down to coincidence, and not because the surrogate model is superior. The ”turns” at low and high loading for the surrogate model can be directly attributed to the divergent behaviour displayed at low and high loading in the plots in figure 13 and 14.

In table 8, the MAPE error of the heat of absorption for the different temperatures are given. The error is comparing the surrogate and the base model. The values are rather reasonable, but also redundant when considering the neural networks performance compared to the eNRTL model.

3.5 Speciation

In order to further ascertain that our model has fully ”understood” the data it has been trained on, the mole fractions in the liquid phase can also be compared to experimental values. The work of Chen et al. (2014)[19] provides us with NMR

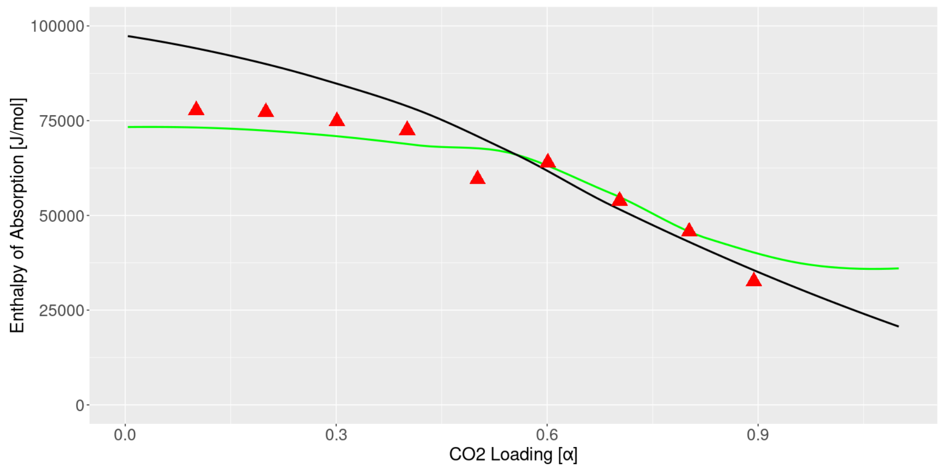


Figure 15: The heat of absorption at 40 degrees plotted against the loading. Green is the data from the surrogate model, black is **base model** data and red is experimental data.

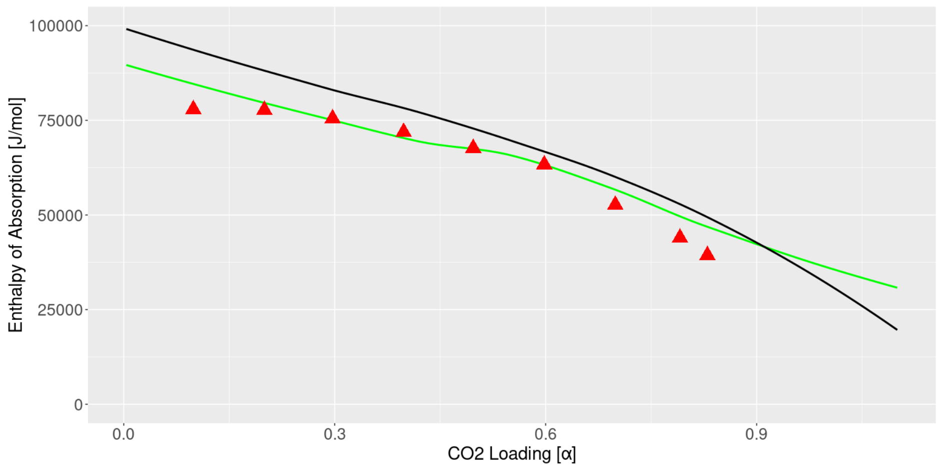


Figure 16: The heat of absorption at 60 degrees plotted against the loading. Green is the data from the surrogate model, black is **base model** data and red is experimental data.

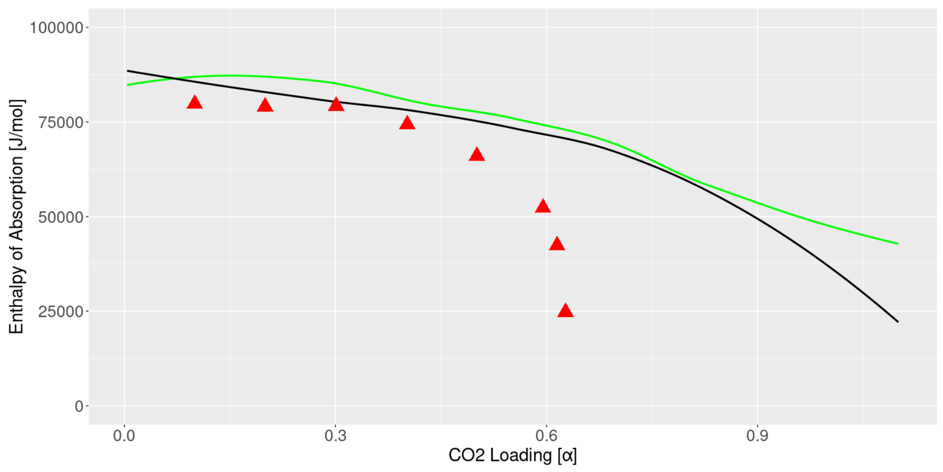
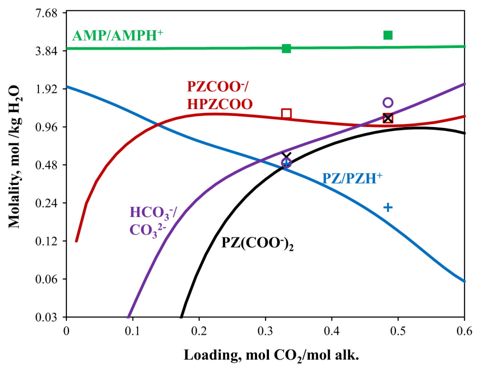


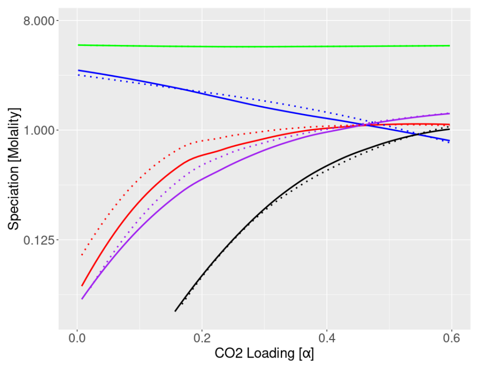
Figure 17: The heat of absorption at 80 degrees plotted against the loading. Green is the data from the surrogate model, black is **base model** data and red is experimental data.

Temperature	MAPE Error [%]
40 degrees	7.2128
60 degrees	6.1222
80 degrees	5.6738

Table 8: A table with the MAPE error of the heat of absorption



(a) Experimental values from Chen et al. (2014)



(b) Values calculated by the model

Figure 18: The molality of selected species plotted against the loading at 40 degrees celsius. The selected species are: $AMP/AMPH^{-}$, $PzCOO^{-}/PzCOOH$, Pz/PzH^{+}, HCO_3^{-} and $Pz(COO^{-})_2$. The solid lines are the neural network model while the dashed are the eNRTL values. The composition is 2m Piperazine and 4m AMP [molality]

speciated data with which we can compare. Unfortunately, their article does not provide the hard data, but rather speciation plots. This means that the comparison won't be as accurate, but it should still be useful

As seen in figure 18, the model does not accurately predict the experimental values from Chen et al. (2014). Some of the predictions are actually quite accurate, but the red prediction along the whole loading interval, and blue and purple across parts of the interval are way off. Now of course, one must consider that only the points in figure 18a are the experimental values derived from NMR spectroscopy. And thus the other values should be taken with a pinch of salt.

3.6 Model training

As mentioned earlier in the section on model training, 1000 epochs was deemed necessary to achieve the desired accuracy. With the 8000 observations in the training set, the model took slighty less than three minutes to train. Each epoch however, took merely 4 milliseconds to propagate. This means that roughly 70% of the time spent training the model is data loading in and out of memory, rather than the actual training itself. Changing the training parametres also does not impact

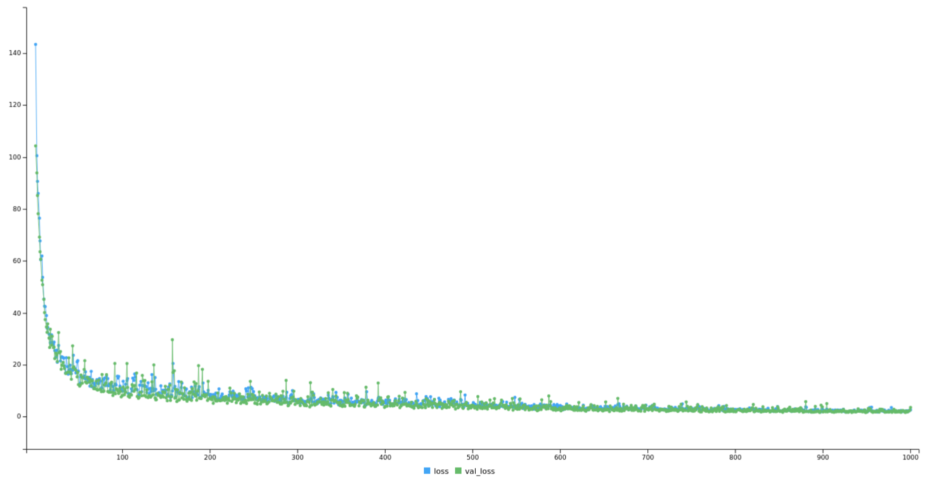


Figure 19: The sum of the cost function plotted against epochs trained. Both the training, and validation error is visible.

the training time in any notable way. When using over 150 perceptrons in the hidden layer, the training time was slightly above 3 minutes, with some training epochs taking 5 ms, with most still within that 4 ms window. This then shows us that the model was very fast, especially since the eNRTL base model, written in matlab, took almost half an hour to do the exact same thing. The goal then, of creating a more computationally efficient model has certainly been reached.

As can be seen in figure 19, the sum of the cost function converges to around 2. And whilst the does not seem to improve after about 200 epochs, it actually does, being around 5 at this point.

3.7 Model assessment

As seen most prominently in figure 13 at around loading = 1.2, and previously discussed, the surrogate model does have these discontinuous ”elbow points”. These discrepancies are normally distributed along low and high loading, and thus should explain the behaviour seen in the error plots in figures 7 to 12. A combination of discontinuous function would not itself be continuous, and this is clearly evident from the figures presented in this chapter. A continuous activation function would approximate the curves of the continuous eNRTL model more faithfully. But as clearly shown in table 5, the continuous activation function consistently give us a higher error than the ReLU function. This then leads us to perhaps the greatest lesson that is to be learned in machine learning or statistical modelling; that building a model is essentially one large compromise subdivided into smaller compromises. What this means in practice is when building a model, you must always choose between accuracy or speed, between interpretability and usability, and perhaps most important, between bias and variance. There is simply no way around it, and for this specific model, the choice was between low error and strange behaviour at the extremities of the loading range.

The surrogate model seems to perform worse at higher temperatures. The reason for this is probably that the dependent variables are more spread out at higher temperatures, and the neural network struggles to approximate all the widely varying values. As can be seen in figures 13 and 14 the growth in partial pressure of CO_2 is almost perfectly exponential. The model was trained on the natural logarithm of the response variables, not the actual values themselves. And for the pressure, which ranges from zero to 300 kPa, a small error in the predicted value may indeed grow large when transformed back into the exponential domain. Of course, this brings us back to the topic of compromises. As the chosen method of transforming the dependent variables gave a larger error at higher temperature, a different way of normalizing these variables might have given us another type of error. An example of this would be the normalization given in equation 30 returning a large consistent error for the pressure, as cramming the substantial span of the pressure between 0 and 1 would probably have an even greater effect when transforming it back into the normal pressure domain.

3.8 Further work

For further work, one should look into training the model on the experimental values in addition to the values from the base model. Conventional logic suggests that training the model on experimental values would improve it's performance with regards to real life performance. However this might prove problematic as there is little data, and the data there is is not evenly distributed. There is therefore no way of knowing how the neural network will react to the added data. Preliminary testing seem to show that the addition of experimental training data does not really benefit the model, and this can probably be explained by the experimental data being poorly distributed along the parametre limits of the model.

There was also some trouble in predicting the heat of absorption at higher temperatures. To make the model predict the heat of absorption with greater accuracy, the heat of absorption should probably be implemented as a dependent variable in it's own right. With it's own thermodynamic model to produce data that the surrogate model can then be trained on.

4 Conclusion

For this specialization project, a surrogate model was created for the purpose of approximating an already existing base model that calculated the vapour/liquid equilibrium of an aqueous AMP/Pz/CO_2 system. The surrogate model was implemented in the tensorflow platform with an R interface. The surrogate model was found to approximate the base model very well at temperatures below 80 degrees celcius, though even at this temperature, the relative errors were mostly within 15%. When calculating the heat of absorption the model also worked remarkably well at 40 and 60 degrees celcius, but quickly fell apart at 80 degrees. The surrogate model exhibited strange ”elbowing” behaviour at the extremes of the loading ranges, this is most likely because the discontinuous ReLU function was used as the activation for the hidden layer. The method of normalizing the independent variables also very probably had an effect on the pressure response. On the other hand, not using these two specific methods would undoubtedly have introduced other issues, like worse accuracy.

The purpose of this specialization project was to create a less computationally complex alternative to the base model, and this was certainly a success. At most input ranges, the surrogate model displayed remarkable accuracy and speed. When comparing against experimental results, the model also performed well enough, though not as well as the base model.

References

[1] Thomas R. Anderson, Ed Hawkins, and Philip D. Jones. Co2, the greenhouse effect and global warming: from the pioneering work of arrhenius and callendar to today's earth system models. *Endeavour*, 40(3):178–187, 2016. ISSN 0160-9327. doi: https://doi.org/10.1016/j.endeavour.2016.07.002. URL `https://www.sciencedirect.com/science/article/pii/S0160932716300308`.

[2] M. Wang, A. Lawal, P. Stephenson, J. Sidders, and C. Ramshaw. Post-combustion co2 capture with chemical absorption: A state-of-the-art review. *Chemical Engineering Research and Design*, 89(9):1609–1624, 2011. ISSN 0263-8762. doi: https://doi.org/10.1016/j.cherd.2010.11.005. URL `https://www.sciencedirect.com/science/article/pii/S0263876210003345`. Special Issue on Carbon Capture & Storage.

[3] Nabil El Hadri, Dang Viet Quang, Earl L.V. Goetheer, and Mohammad R.M. Abu Zahra. Aqueous amine solution characterization for post-combustion co2 capture process. *Applied Energy*, 185:1433–1449, 2017. ISSN 0306-2619. doi: https://doi.org/10.1016/j.apenergy.2016.03.043. URL `https://www.sciencedirect.com/science/article/pii/S0306261916303609`. Clean, Efficient and Affordable Energy for a Sustainable Future.

[4] Ardi Hartono, Rafiq Ahmad, Muhammad Usman, Naveed Asif, and Hallvard F. Svendsen. Solubility of co2 in 0.1m, 1m and 3m of 2-amino-2-methyl-1-propanol (amp) from 313 to 393k and model representation using the enrtl framework. *Fluid Phase Equilibria*, 511:112485, 2020. ISSN 0378-3812. doi: https://doi.org/10.1016/j.fluid.2020.112485. URL `https://www.sciencedirect.com/science/article/pii/S0378381220300315`.

[5] Gary T. Rochelle. Amine scrubbing for co2 capture. *Science*, 325(5948):1652–1654, 2009. ISSN 0036-8075. doi: 10.1126/science.1176731. URL `https://science.sciencemag.org/content/325/5948/1652`.

[6] James Yeh, Henry Pennline, and Kevin Resnik. Study of co 2 absorption and desorption in a packed column. *Energy & Fuels - ENERGY FUEL*, 15:274–278, 03 2001. doi: 10.1021/ef0002389.

[7] Fei Li, Jie Zhang, Chao Shang, Dexian Huang, Eni Oko, and Mei-

hong Wang. Modelling of a post-combustion co2 capture process using deep belief network. *Applied Thermal Engineering*, 130:997–1003, 2018. ISSN 1359-4311. doi: https://doi.org/10.1016/j.applthermaleng.2017.11.078. URL `https://www.sciencedirect.com/science/article/pii/S1359431116344040`.

[8] J. H. Friedman, R. Tibshirani, and T. Hastie. *Elements of Statistical Learning*. Springer, 1st edition, 2001.

[9] Griet Heuvelmans, Bart Muys, and Jan Feyen. Regionalisation of the parameters of a hydrological model: Comparison of linear regression models with artificial neural nets. *Journal of Hydrology*, 319(1):245–265, 2006. ISSN 0022-1694. doi: https://doi.org/10.1016/j.jhydrol.2005.07.030. URL `https://www.sciencedirect.com/science/article/pii/S0022169405003641`.

[10] Gitta Kutyniok. Discussion of: "Nonparametric regression using deep neural networks with ReLU activation function". *The Annals of Statistics*, 48(4): 1902 – 1905, 2020. doi: 10.1214/19-AOS1911. URL `https://doi.org/10.1214/19-AOS1911`.

[11] Ciaburro and Venkateswaran. *Neural Networks in R*. Packt Publishing, 1st edition, 2017.

[12] D. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *ICLR 2015*, 2015.

[13] Hallvard F. Svendsen, Erik T. Hessen, and Thor Mejdell. Carbon dioxide capture by absorption, challenges and possibilities. *Chemical Engineering Journal*, 171(3):718–724, 2011. ISSN 1385-8947. doi: https://doi.org/10.1016/j.cej.2011.01.014. URL `https://www.sciencedirect.com/science/article/pii/S1385894711000416`. Special Section: Symposium on Post-Combustion Carbon Dioxide Capture.

[14] Chau-Chyun Chen and L. B. Evans. A local composition model for the excess gibbs energy of aqueous electrolyte systems. *AIChE Journal*, 32(3):444–454, 1986. doi: https://doi.org/10.1002/aic.690320311. URL `https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690320311`.

[15] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Control Signal Systems*, 2:303–314, 1989. doi: https://doi.org/10.1007/

BF02551274. URL https://link.springer.com/article/10.1007/BF02551274#citeas.

[16] Carl Friederich Gauss. *Theoria combinationis observationum erroribus minimis obnoxiae*. 1823.

[17] Sukanta Kumar Dash, Arunkumar Samanta, Amar Nath Samanta, and Syamalendu S. Bandyopadhyay. Vapour liquid equilibria of carbon dioxide in dilute and concentrated aqueous solutions of piperazine at low to high pressure. *Fluid Phase Equilibria*, 300(1):145–154, 2011. ISSN 0378-3812. doi: https://doi.org/10.1016/j.fluid.2010.11.004. URL https://www.sciencedirect.com/science/article/pii/S0378381210005546.

[18] Qian Xie, Adisorn Aroonwilas, and Amornvadee Veawab. Measurement of heat of co2 absorption into 2-amino-2-methyl-1-propanol (amp)/piperazine (pz) blends using differential reaction calorimeter. *Energy Procedia*, 37:826–833, 2013. ISSN 1876-6102. doi: https://doi.org/10.1016/j.egypro.2013.05.175. URL https://www.sciencedirect.com/science/article/pii/S1876610213001859. GHGT-11 Proceedings of the 11th International Conference on Greenhouse Gas Control Technologies, 18-22 November 2012, Kyoto, Japan.

[19] Han Li, Peter T. Frailie, Gary T. Rochelle, and Jian Chen. Thermodynamic modeling of piperazine/2-aminomethylpropanol/co2/water. *Chemical Engineering Science*, 117:331–341, 2014. ISSN 0009-2509. doi: https://doi.org/10.1016/j.ces.2014.06.026. URL https://www.sciencedirect.com/science/article/pii/S0009250914003133.

A Appendix

The model was written in R with the calculations themselves being implemented in python by the Tensorflow library. The code in section A.1 was always used to create the model, whilst the codes in sections A.2 to A.5 was appended to this original code and executed with it. The training and subsequent data sets were produced by the base eNRTL model of Hartono et al in Matlab.

A.1 Training the neural network

This is the code for training the surrogate model.

```


#Reading the .csv files generated by the Matlab model
input <- as.matrix(read.csv(file="AmpIn12.csv"))
output <- as.matrix(read.csv(file="AmpOut12.csv"))

#Sorting out the outliers, as they are of little interest to our
    case
InputOutput <- cbind(input, output)
InputOutput <- as.data.frame(subset(InputOutput, InputOutput
   [,24] <= 300))


input <- as.data.frame(rbind(InputOutput[,1:4]))
output <- as.data.frame(rbind(InputOutput[,5:25]))



#loading the required libraries
library(keras)
library(mlbench)
library(dplyr)
library(magrittr)
library(neuralnet)
library(tensorflow)
library(ggplot2)


#Putting everything back into a dataframe and naming the columns
    .
data <- cbind(input, output)
colnames(data) <- c("alpha", "wamin1", "wamin2", "Temp", "x1", "
    x2", "x3", "x4", "x5", "x6", "x7", "x8", "x9", "x10", "x11",
    "x12", "x13", "x14", "x15", "y1", "y2", "y3", "y4", "Pt", "I"
    )


#Splitting data into a training and a validation set, and
    transforming them into the appropriate data type.
dt = sort(sample(nrow(data), nrow(data)*.8))
train<-data[dt,]
test<-data[-dt,]



training <- train[1:4]
testing <- test[1:4]
trainingtarget <- train[5:24]
testingtarget <- test[5:24]

x.train <- training
x.test <- testing

x.train <- as.matrix(x.train)
x.test <- as.matrix(x.test)

y.train <- as.matrix(trainingtarget)
y.test <- as.matrix(testingtarget)

#Scaling the input variables from 0 -> 1.
#The network converges easier and more accurately if the
#variables are in the same domain.

mean <- apply(x.train, 2, mean)
std <- apply(x.train, 2, sd)
x.train <- as.array(scale(x.train, center = mean, scale = std))
x.test <- as.array(scale(x.test, center = mean, scale = std))



#Creating the model itself
# This version has l1 regularization
model <-keras_model_sequential()%>%
  layer_dense(units = 110, activation = "relu", input_shape =c
    (4), kernel_regularizer = regularizer_l1(l = 0.0005))%>%
  layer_dense(units = 20, activation = "linear")

```

```
model%>%
  compile(optimizer = "adam", loss = "mape")

#Starting the training of the model
history <- model%>%
  fit(x.train , y.train , epochs = 1000, batch_size = 120,
  validation_split = 0.2)


#Evaluating the testing set
#and transforming the predicted set back from the log scale

model %>% evaluate(x.test , y.test)
pred1 <- model %>% predict(x.test)

pred <- exp(pred1)
y.test <- exp(y.test)



#Finding Mean Absolute Percentage Error
#The errors are the MAPE error even if they are labelled mse
mse1 <- (((as.vector(pred[,1]) -as.vector(y.test[,1]))/as.vector
    (pred[,1])))
mse2 <- abs(mean((as.vector(pred[,2]) -as.vector(y.test[,2]))/as
    .vector(pred[,2])))
mse3 <- abs(mean((as.vector(pred[,3]) -as.vector(y.test[,3]))/as
    .vector(pred[,3])))
mse4 <- abs(mean((as.vector(pred[,4]) -as.vector(y.test[,4]))/as
    .vector(pred[,4])))
mse5 <- abs(mean((as.vector(pred[,5]) -as.vector(y.test[,5]))/as
    .vector(pred[,5])))
mse6 <- abs(mean((as.vector(pred[,6]) -as.vector(y.test[,6]))/as
    .vector(pred[,6])))
mse7 <- abs(mean((as.vector(pred[,7]) -as.vector(y.test[,7]))/as
    .vector(pred[,7])))
mse8 <- abs(mean((as.vector(pred[,8]) -as.vector(y.test[,8]))/as
    .vector(pred[,8])))
mse9 <- abs(mean((as.vector(pred[,9]) -as.vector(y.test[,9]))/as
    .vector(pred[,9])))
mse10 <- abs(mean((as.vector(pred[,10]) -as.vector(y.test[,10]))
    /as.vector(pred[,10])))
mse11 <- abs(mean((as.vector(pred[,11]) -as.vector(y.test[,11]))
    /as.vector(pred[,11])))
mse12 <- abs(mean((as.vector(pred[,12]) -as.vector(y.test[,12]))
    /as.vector(pred[,12])))
mse13 <- abs(mean((as.vector(pred[,13]) -as.vector(y.test[,13]))
    /as.vector(pred[,13])))
mse14 <- abs(mean((as.vector(pred[,14]) -as.vector(y.test[,14]))
    /as.vector(pred[,14])))
mse15 <- abs(mean((as.vector(pred[,15]) -as.vector(y.test[,15]))
    /as.vector(pred[,15])))
mse16 <- (((as.vector(pred[,16]) -as.vector(y.test[,16]))/as.
    vector(pred[,16])))
mse17 <- abs(mean((as.vector(pred[,17]) -as.vector(y.test[,17]))
    /as.vector(pred[,17])))
mse18 <- abs(mean((as.vector(pred[,18]) -as.vector(y.test[,18]))
    /as.vector(pred[,18])))
mse19 <- abs(mean((as.vector(pred[,19]) -as.vector(y.test[,19]))
    /as.vector(pred[,19])))
mse20 <- (((as.vector(pred[,20]) -as.vector(y.test[,20]))/as.
    vector(pred[,20])))



#Adding the different MAPE errors to the same vector
MSE <- cbind(mse1, mse2, mse3, mse4, mse5, mse6, mse7, mse8,
    mse9, mse10, mse11, mse12, mse13, mse14, mse15, mse16, mse17,
     mse18, mse19, mse20)


#Giving the vector elements names
colnames(MSE) <- c("x1 mape", "x2 mape", "x3 mape", "x4 mape", "
    x5 mape", "x6 mape", "x7 mape", "x8 mape", "x9 mape", "x10
    mape", "x11 mape", "x12 mape", "x13 mape", "x14 mape", "x15
    mape", "y1 mape", "y2 mape", "y3 mape", "y4 mape", "Pt mape")

#Printing the MAPE vector
MSE


#Calculating and printing the mean of the MAPE vector
MAPE <- mean(MSE)
MAPE
```

A.2 Plotting the residual and relative error

```
#Reading in the evaluation data
EvalIn <- as.matrix(read.csv(file="ResidualIn.csv"))
EvalOut <- as.matrix(read.csv(file="ResidualOut.csv"))

EvalInputOutput <- cbind(EvalIn, EvalOut)
EvalInputOutput <- as.data.frame(subset(EvalInputOutput,
    EvalInputOutput[,24] <= 300))


EvalIn <- as.data.frame(rbind(EvalInputOutput[,1:4]))
EvalOut <- as.data.frame(rbind(EvalInputOutput[,5:25]))

#Scaling the new data using the distribution of the training
    dataset
x.eval <- scale(EvalIn, center = mean, scale = std)
y.eval <- EvalOut


#Propagating the new data through the neural network

pred_new <- predict(model, x.eval)
pred_new <- as.matrix(exp(pred_new))


#calculating the residual errors
mse16 <- (((as.vector(pred_new[,16]) -as.vector(y.eval[,16]))))
mse20 <- (((as.vector(pred_new[,20]) -as.vector(y.eval[,20]))))


#For the residual error these two lines of code would not be
    needed as they are only used to find the relative error
mse16 <- mse16/pred_new[,16]
mse20 <- mse20/pred_new[,20]



#Plotting the errors of yCO2 and the total pressure
ggplot() +
  geom_point(aes(x=EvalIn[,1], y = mse16), color = "blue") +
  xlim(0,1) +
  ylim(-0.3, 0.3) +
  xlab("CO2 Loading [alpha]") +
  ylab("Residual Error of yCO2 @ 60 degrees") +
  theme(axis.title = element_text(size = rel(2)),
         axis.text = element_text(size = 20))


ggplot() +
  geom_point(aes(x=EvalIn[,1], y = mse20), color = "blue") +
  xlim(0,1) +
  ylim(-100, 100) +
  xlab("CO2 Loading [alpha]") +
  ylab("Residual Error of Pressure @ 60 degrees") +
  theme(axis.title = element_text(size = rel(2)),
         axis.text = element_text(size = 20))
```

A.3 Calculating and plotting the heat of absorption

```

#Reading in the data
EvalIn <- as.matrix(read.csv(file="HeatAbsIn_d.csv"))
EvalOut <- as.matrix(read.csv(file = "HeatAbsOut_d.csv"))


#Adding the dt step to the temperature for numerical
    differentiation
EvalIn2 <- EvalIn[,4]+0.0001

Heat1 <- EvalIn
Heat2 <- cbind(EvalIn[,1:3], EvalIn2)

#Normalizing the matrices
Heat1_2 <- as.array(scale(Heat1, center = mean, scale = std))
Heat2_2 <- as.array(scale(Heat2, center = mean, scale = std))

Eval1 <- exp(model %>% predict(Heat1_2))
Eval2 <- exp(model %>% predict(Heat2_2))

#Calculating the heat of absorption
HeatAbs <- -((log(Eval2[,17]*Eval2[,20])-log(Eval1[,17]*Eval1
    [,20]))/(1/(Heat2[,4])-1/Heat1[,4]))*8.314



#Plotting in the experimental values
alpha <- c(0.101, 0.2, 0.301, 0.401, 0.501, 0.601, 0.703, 0.802,
     0.894)
habs <- c(77.77, 77.26, 74.86, 72.52, 59.54, 63.95, 53.87,
    45.71, 32.61)*1000

#The sections that are commented out are for the other
    temperatures, this was done one temperature at the time

#alpha <- c(0.099, 0.2, 0.297, 0.398, 0.497, 0.598, 0.699,
    0.791, 0.830)
#habs <- c(77.89, 77.71, 75.47, 71.89, 67.63, 63.32, 52.65,
    43.95, 39.34)*1000

#alpha <- c(0.1, 0.2, 0.301, 0.402, 0.501, 0.595, 0.615, 0.627)
#habs <- c(79.78, 79.09, 79.26, 74.36, 66.04, 52.32, 42.35,
    24.79)*1000

```

```

#Importing the model from matlab for the purposes of finding the
    error
#The .csv files ending in _dt have the dt step for numerical
    differentiation added as in line 8
MatlabIn <- as.matrix(read.csv(file="HeatAbsIn_d.csv"))
MatlabOut <- as.matrix(read.csv(file="HeatAbsOut_d.csv"))
MatlabIn2 <- as.matrix(read.csv(file="HeatAbsIn_dt.csv"))
MatlabOut2 <- as.matrix(read.csv(file="HeatAbsOut_dt.csv"))

Finding the heat of absorption of the base model
HeatAbs2 <- -((log(MatlabOut2[,17]*MatlabOut2[,20])-log(
    MatlabOut[,17]*MatlabOut[,20]))/(1/(MatlabIn2[,4])-1/MatlabIn
    [,4]))*8.314


#Calculating the error between the surrogate and base model
error <-  abs(mean((as.vector(HeatAbs) -as.vector(HeatAbs2))/as.
    vector(HeatAbs)))
error

#Plotting the values against each other
ggplot() +
  geom_smooth(aes(x=EvalIn[,1], y=HeatAbs), color = "green", se=
      FALSE) +
  geom_smooth(aes(x=MatlabIn[,1], y=HeatAbs2), color = "black",
      se=FALSE) +
  geom_point(aes(x=alpha, y = habs), color = "red", shape = 17,
      size = 6) +
  ylim(0, 100000) +
  xlab("CO2 Loading [alpha]") +
  ylab("Enthalpy of Absorption [J/mol]") +
  theme(axis.title = element_text(size = rel(2)),
        axis.text = element_text(size = 20))
```

A.4 Plotting the partial pressure of CO2 against the loading

```r

#Reading evaluation data
EvalIn <- as.matrix(read.csv(file="ExperPlotIn4.csv"))
EvalOut <- (read.csv(file = "ExperPlotOut4.csv"))

#Scaling the new data using the distribution of the training
    dataset
x.eval <- scale(EvalIn, center = mean, scale = std)
y.eval <- EvalOut



#Evaluating and scaling back to original domain
pred_new <- predict(model, x.eval)
pred_new <- as.matrix(exp(pred_new))


y1 <- pred_new[,20]*pred_new[,17]
y2 <- EvalOut[,20]*EvalOut[,17]


#Experimental data
alpha <- c(0.34, 0.407, 0.51, 0.606, 0.725, 0.784, 0.835, 0.869,
     0.931, 0.964, 0.983, 1.007, 1.021)
p <- (c(0.17, 0.533, 1.09, 3.57, 7.43, 18.32, 53.49, 93.97,
    231.7, 538.1, 755.2, 1059, 1393))


#Plotting the partial pressure against the loading
ggplot() +
  geom_point(aes(x=alpha, y=p), color = "blue") +
  geom_line(aes(x=EvalIn[,1], y=(y1)), color = "green") +
  geom_line(aes(x=EvalIn[,1], y=(y2)), color = "red") +
  xlab("Loading [alpha] ") +
  ylab("Partial Pressure of CO2 [Pa]") +
  scale_y_log10(limits = c(1e-3,1e4)) +
  theme(axis.title = element_text(size = rel(2)),
        axis.text = element_text(size = 20))
```

A.5 Plotting the speciation

```

#Loading in the data
SpeciationIn <- read.csv("SpeciationIn.csv")
SpeciationOut <- read.csv("SpeciationOut.csv")

#Scaling the new data along the same parametres as the training
    set
Speciation <- as.array(scale(SpeciationIn, center = mean, scale
    = std))


#propagating the new data through the model and scaling it back
    to the original domain
Speciation1 <- exp(model %>% predict(Speciation))

#Finding the molality of the different species

#Molality of variables
m1 <- (Speciation1[,4]+Speciation1[,7])/(Speciation1[,1]*18/
    1000)
m2 <- (Speciation1[,12]+Speciation1[,15])/(Speciation1[,1]*18/
    1000)
m3 <- (Speciation1[,3]+Speciation1[,6])/(Speciation1[,1]*18/
    1000)
m4 <- (Speciation1[,10]+Speciation1[,11])/(Speciation1[,1]*18/
    1000)
m5 <- (Speciation1[,13])/(Speciation1[,1]*18/1000)

#Molality of eNRTL values
m6 <- (SpeciationOut[,4]+SpeciationOut[,7])/(SpeciationOut[,1]*
    18/1000)
m7 <- (SpeciationOut[,12]+SpeciationOut[,15])/(SpeciationOut[,1]
    *18/1000)
m8 <- (SpeciationOut[,3]+SpeciationOut[,6])/(SpeciationOut[,1]*
    18/1000)
m9 <- (SpeciationOut[,10]+SpeciationOut[,11])/(SpeciationOut[,1]
    *18/1000)
m10 <- (SpeciationOut[,13])/(SpeciationOut[,1]*18/1000)



#Plotting the different species against the loading
ggplot() +
geom_smooth(aes(x=SpeciationIn[,1], y=m1), color = "green", se
    =FALSE) +
geom_smooth(aes(x=SpeciationIn[,1], y=m2), color = "red", se=
    FALSE) +
geom_smooth(aes(x=SpeciationIn[,1], y=m3), color = "blue", se=
    FALSE) +
geom_smooth(aes(x=SpeciationIn[,1], y=m4), color = "purple",
    se=FALSE) +
geom_smooth(aes(x=SpeciationIn[,1], y=m5), color = "black", se
    =FALSE) +
geom_smooth(aes(x=SpeciationIn[,1], y=m6), color = "green", se
    =FALSE, linetype = "dotted") +
geom_smooth(aes(x=SpeciationIn[,1], y=m7), color = "red", se=
    FALSE, linetype = "dotted") +
geom_smooth(aes(x=SpeciationIn[,1], y=m8), color = "blue", se=
    FALSE, linetype = "dotted") +
geom_smooth(aes(x=SpeciationIn[,1], y=m9), color = "purple",
    se=FALSE, linetype = "dotted") +
geom_smooth(aes(x=SpeciationIn[,1], y=m10), color = "black",
    se=FALSE, linetype = "dotted") +
xlab("CO2 Loading [alpha]") +
ylab("Speciation [Molality]") +
scale_y_continuous(trans = "log2",limits = c(0.03,7.68)) +
theme(axis.title = element_text(size = rel(2)),
      axis.text = element_text(size = 20)) +
xlim(0, 0.6)
```

A.6 Calculating the surrogate model's deviation from experimental values

This specific code was used to find the errors for the plots in figure 13, but the calculation for all other experimental data points is essentially the same.

```


#Importing the experimental values
alpha <- c(0.1895, 0.2103, 0.2887, 0.314, 0.378, 0.4351, 0.4666,
    0.5166, 0.5792, 0.5992, 0.6085, 0.6167, 0.6559, 0.6976,
    0.7285, 0.7337)
p <- (c(0.026, 0.034, 0.09, 0.113, 0.201, 0.377, 0.534, 0.947,
    1.598, 2.308, 3.039, 3.06, 5.182, 10.117, 13.323, 22.142))

alpha2 <- c(0.0912, 0.117, 0.1662, 0.2072, 0.2412, 0.2706,
    0.2998, 0.3287, 0.4245, 0.484, 0.5252, 0.5417, 0.5902, 0.613)
p2 <- (c(0.044, 0.075, 0.165, 0.269, 0.402, 0.54, 0.809, 0.917,
    2.346, 4.617, 6.777, 8.161, 11.455, 19.655))

alpha3 <- c(0.0368, 0.0346, 0.0542, 0.0817, 0.0982, 0.1143,
    0.1411, 0.1907, 0.229, 0.2615, 0.2991, 0.3252, 0.4175)
p3 <- c(0.036, 0.059, 0.106, 0.238, 0.361, 0.482, 0.708, 1.362,
    2.209, 3.244, 4.254, 5.243, 13.064)


#Importing in datasets
EvalIn1 <- as.matrix(read.csv(file="ExperErrorIn.csv"))
EvalIn2 <- as.matrix(read.csv(file="ExperErrorIn2.csv"))
EvalIn3 <- as.matrix(read.csv(file="ExperErrorIn3.csv"))

x.eval1 <- scale(EvalIn1, center = mean, scale = std)
x.eval2 <- scale(EvalIn2, center = mean, scale = std)
x.eval3 <- scale(EvalIn3, center = mean, scale = std)


pred_new1 <- exp(predict(model, x.eval1))
pred_new2 <- exp(predict(model, x.eval2))
pred_new3 <- exp(predict(model, x.eval3))



#Finding the partial pressure of CO2
y1 <- pred_new1[,17]*pred_new1[,20]
y2 <- pred_new2[,17]*pred_new2[,20]
```

```
y3 <- pred_new3[,17]*pred_new3[,20]

#Finding the errors
mape1 <- abs((y1-p)/p)
mape2 <- abs((y2-p2)/p2)
mape3 <- abs((y3-p3)/p3)

mean(mape1)
mean(mape2)
mean(mape3)
mape1
mape2
mape3



 runif(3)
```