CHALMERS
UNIVERSITY OF TECHNOLOGY

CREATING A DIGITAL CLONE OF A PROCESS PLANT USING NEURAL NETWORKS

Study case of an Integrated Gasification Combined Cycle plant

In cooperation with AFRY

Master's Thesis in Sustainable Energy Systems

Renesteban Forero Franco

DEPARTMENT OF SPACE, EARTH AND ENVIRONMENT

CHALMERS UNIVERSITY OF TECHNOLOGY

GOTHENBURG, 2020

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

Abstract

Process industry is challenged by complex and highly non-linear systems that requires the use of first principles modelling tools to predict their behavior in relation to certain input conditions. Such modelling tools struggle with convergence issues, local optima traps, and other calculation problems, making any optimization and forecasting estimation process time consuming. This work investigates the efficiency of a Deep Neural Network (DNN) surrogate model in relation to a first principle model. The DNN simulates an Integrated Gasification Combined Cycle plant (IGCC) and compares accuracy and time efficiency in relation to the original Aspen Plus® model. A hyperparameters tuning process is followed to find the optimal network architecture and minimum number of samples that are needed to describe the simulated system. The developed Neural Network model is about 3 orders of magnitude more time efficient in calculations with an accuracy comparable to the Aspen model. Therefore, the Deep Neural Networks is concluded a promising tool for real plant operation in the cost-optimal decision-making process.

1. INTRODUCTION

From the beginning of the first industrial revolution, one of the most important interests has been the ability to model physical systems to predict outputs given certain conditions. This issue has been a matter of high relevance as a means to look for more energy-efficient processes, especially under the urgency of the climate change concern. Modern chemical process plants involve a substantial number of unit operations which commonly are designed to be highly energy integrated and to get the maximum efficiency on its entire cycle. This design work is a combination of physics and chemistry knowledge, process plant experience, simulation schemes and, to some extent, trial and error. However, even with highly skilled engineers and the most advanced simulation tools, navigating the sea of operating points is not an easy task, especially when dealing with highly non-linear and multivariable spaces.

Later on, during operation, external or internal factors require the searching "on the go" of optimal operating points for the process. Price fluctuations of the feed streams or processed products, changes in the required product specs or waste-handling costs, and other factors, force plant management to search for other adjusted cost-optimal conditions under which to operate safely and efficiently. Therefore, the intuition coming from a plant specific experience plays more than often a vital role in the process modeling and results evaluation.

Classical simulation tools based on first principles are the workhorse of many applications, especially when describing most of the chemical processes and its non-linear behavior. However, any simulation of a non-linear model tends to be very time and computationally intensive and normally depends heavily on the starting conditions to avoid convergence problems. In that sense, the pursuit of getting better and faster modeling tools to find optimum operation points, is an issue that continues to be the protagonist in the timeline of the industrial development.

Artificial Neural Networks (ANNs) have found plenty application in the characterization of highly non-linear and complex systems. This ability is especially useful when modeling systems with a relatively high number of parameters and processes as the ones that are normally found in the industrial process plants. One of the branches within ANN's is the so called Deep Neural Networks (DNNs). This is characterized when a deeper level of complexity is added to the network (more than one layer), allowing to go a further step in the description of very complex systems. Several industry sectors have started to look over the implementation of Neural Network algorithms within their processes, due to the significant advantage that represents its use in terms of versatility and robustness to predict behaviors that otherwise would be very hard and time-consuming to predict by the conventional modeling tools.

This thesis project discusses the application of ANNs in process engineering as a tool to mimic the impact of the different variables often manipulated during a process plant operation. The spine cord of the work will be the implementation and results evaluation of Deep Neural Networks upon a conventional simulation model in Aspen Plus of an Integrated Gasification Combined Cycle Plant (IGCC) for generating power, methane, and ammonia.

1.1. Objectives

The present work aims to evaluate the efficiency of a deep neural network algorithm as a surrogate model relative to a state of the art process simulation model. The DNN will be evaluated in terms of accuracy and time efficiency in relation to the traditional modelling tool Aspen Plus. The accuracy of the DNN is evaluated against the Aspen model of a process plant, i.e. the objective of the DNN is to mimic the simulation model. The time efficiency is checked by comparing the calculation speed of the DNN relative to the Aspen model in a quick optimization task.

Specifically, the work aims to:

- Determine the most appropriate method and route to generate the data from the Aspen model in order to deal with convergence errors problems.
- Implement a Deep Neural Network as a surrogate model upon the process plant, and perform an assessment in terms of optimal hyperparameters, and amount of data requirements.
- Assess the performance of the Neural Network model by determining its accuracy and simulation times with respect of the Aspen model.

1.2. Project Limitations

A detailed simulation that considers all the components and sub-processes present within the system, will result in a relatively long simulation time per input set. Thus, given the limitations in available time and computational resources, a constraint was imposed upon the number of unit operations and process blocks that are present in the considered process.

There is a significant number of parameters that play a role in the plant model operation. In order to reduce the degrees of freedom in the input space, only a selected set of parameters was chosen heuristically. The non-chosen parameters were kept fixed with values that are considered as standard for the kind of plant that is being studied.

A full optimization study of the system is not a part of the present research work, but a discussion and a first approximation description over the possible optimization methods will be presented as a matter to be considered in a future research.

2. BACKGROUND

This study focuses on the implementation of a Neural Network over a simulated Integrated Gasification Combined Cycle (IGCC) process plant. Therefore, the two main subjects upon which the study was based are the simulation model of an IGCC plant in a traditional modeling tool as Aspen Plus and the Deep Neural Networks model. The IGCC plant was selected because represents a highly non-linear system with several units and coupled processes, which makes it perfectly suitable to prove the Neural Networks concept.

2.1. IGCC Process Overview

Energy in today's world is a commodity with a dramatically increased demand. Although several kinds of renewables energies have been introduced in the energy supply scheme, coal-based energy generation still continue to have the biggest share in the worldwide picture. However, the burning of coal to generate energy, comes with a huge environmental cost. The finding of solutions to reduce that impact is one of the main quests of the energy engineering research field.

Since the 1920's a process called "Gasification" has been used as one of the methods for the coal energy extraction. One of the technologies that uses it as a front-end process is the IGCC, and represents a very good approach to solve the impact of CO2 emissions. One of the main reasons for this is that the IGCC has the flexibility to incorporate CO_2 capture processes before the synthesis gas be used as a power generation fuel (reference [1], chapter 5). A typical IGCC facility comprises different units and processes to harness the products of the coal or biomass gasification. In Figure 1 It is shown the schematic process.

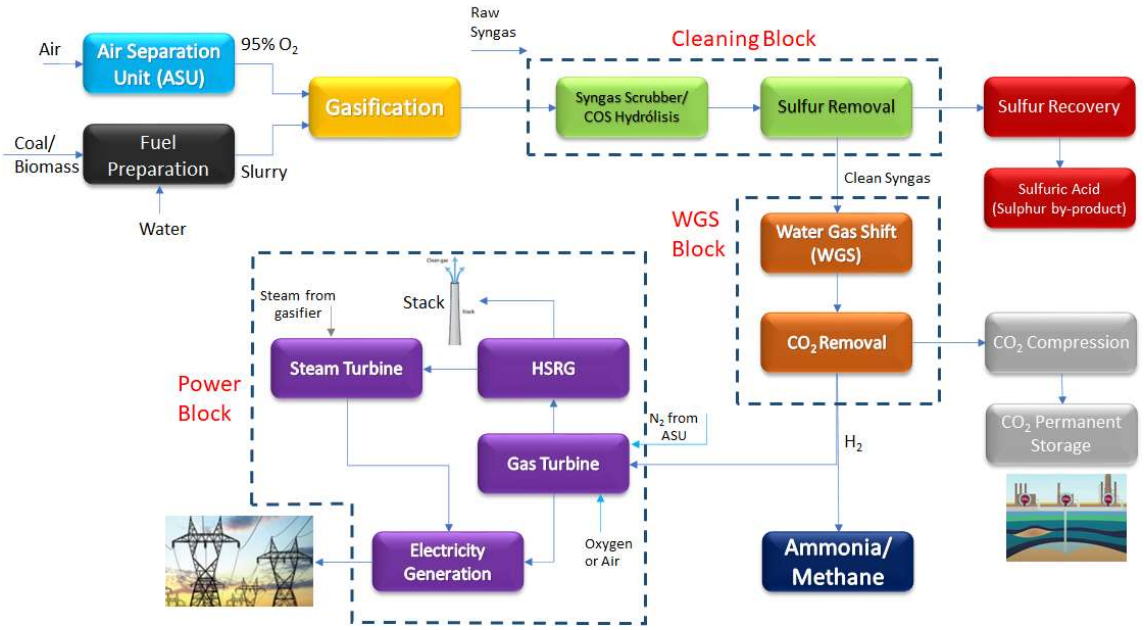


Figure 1. Schematic of an IGCC plant process.

A common IGCC process consists of 6 big blocks: The fuel preparation block, the air Separation Unit, the Gasification block, Cleaning system, Water Gas Shift (WGS) and the Power block.

The process starts with coal entering the system in the Coal preparation block, although biomass can also be incorporated in the stream. There, the fuel stream mixes with water to create a slurry that has similar consistency of the heavy oil. Parallel to this block, a highly concentrated oxygen stream (~95%wt) is generated in the Air Separation Unit which also creates a pure Nitrogen stream (~98%wt). The slurry and oxygen streams are therefore injected into the gasifier (Figure 2) where partial oxidation occurs over a poor oxygen atmosphere at high pressure.

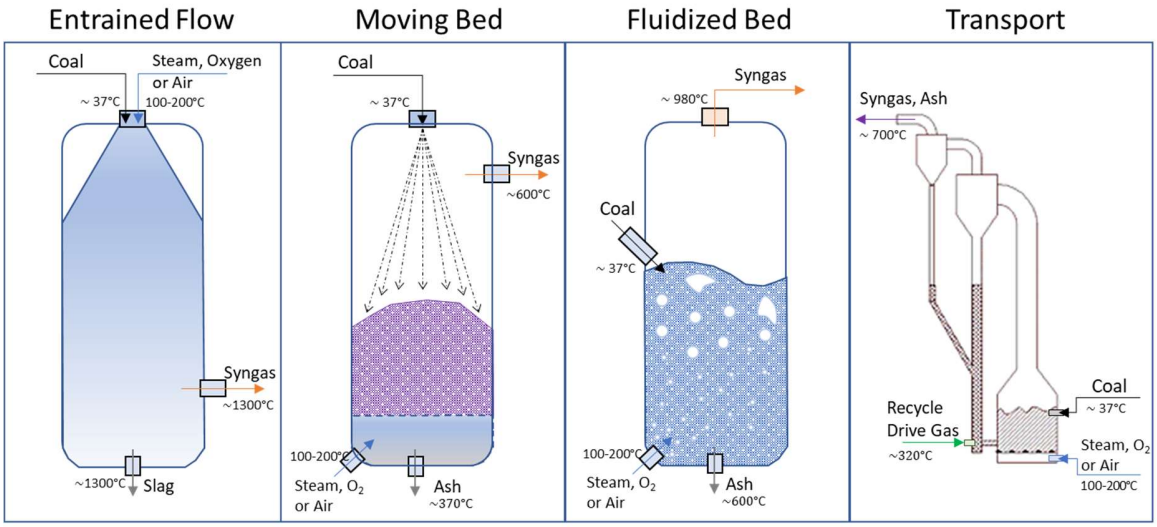


Figure 2. Common types of Gasifiers found in an IGCC plant.

The products of the oxidation are mainly: Syngas, which is a combination of hydrogen (H_2) and carbon monoxide (CO), hydrogen sulfide (H_2S), other carbon sulfide gases, and heat. The gases go directly to the cleaning section while the heat energy is harnessed to create high pressure steam for turbines to electric power generation or other applications. If the fuel is represented simply as carbon then the main reactions that take place in the gasifier can be simplified as described in equation (1). (see reference [1] and [2])

Name of reaction	**Reaction**
Incomplete oxidation	$C + {}^{1}/_{2}\, O_2 \rightarrow CO$
Oxidation	$C + O_2 \rightarrow CO_2$
Water gas	$C + H_2O \rightarrow CO + H_2$
Boudouard	$C + CO_2 \rightarrow 2CO$
Shift	$CO + H_2O \rightarrow CO_2 + H_2$
Hydrogasification	$C + 2H_2 \rightarrow CH_4$
Methanation	$CH_4 + H_2O \rightarrow CO + 3H_2$
Ammonia formation	$N_2 + 3H_2 \rightarrow 2NH_3$
Hydrogen sulfide	$H_2 + S \rightarrow H_2S$

(1)

In the Cleaning section, the sulfur gases are stripped out from the main stream, and are later treated for sulfur extraction. After this, the almost pure syngas is sent to the water gas shift section, where it is combined with high pressure steam, to make react the CO, and get more H_2 and CO_2 in a water shift reactor. The CO_2 is latter removed in a scrubber using methanol (MeOH), and can finally be stored in permanent geological deposits. The concentrated H_2 stream can be used for electric power

generation in a Combined Cycle Gas Turbine plant (CCGT). Variations can occur in the process, where the syngas can directly be used for power generation by making it react with more oxygen to finishing the CO oxidation. Alternatively, the syngas can also be used to synthesize Methane gas by the Sabatier Reaction. Also, the Hydrogen can be combined con a Nitrogen stream to produce ammonia, by the means of the Haber-Bosch process, which is highly demanded in the urea and fertilizers industry. A more detailed explanation about the different kinds of processes, components and main existing plants worldwide can be found in reference [1].

2.2. Neural Networks Overview

2.2.1. Deep Neural Networks as a Concept

During the last two decades, the field of Artificial Intelligence has seen an exponential development and inclusion in many areas of society. Although this field started about 60 years ago as a mere computational concept, in the last years it has become one of the main leading forces in the transition process to a new industrial era. One of the main subsets of this big field is the one commonly known as Machine Learning, where special statistical algorithms are implemented to learn from a structured data set to make predictions about something [3]. Making a step forward into the Machine Learning field, it is possible to distinguish one of the most fundamental areas that have opened the door to unnumbered kinds of applications and that is the Deep Neural Networks. This concept is based on the attempt to emulate the way the human brain processes the incoming information from the external world to make predictions and take decisions. Highly interconnected nodes forming an arrangement of several successive layers resemble what the Neurons carry out in a brain: learning and capturing patterns from the incoming data to generate outputs based on activation signals. This human brain emulation is the reason why this computation homologous is normally called Artificial Neural Networks.

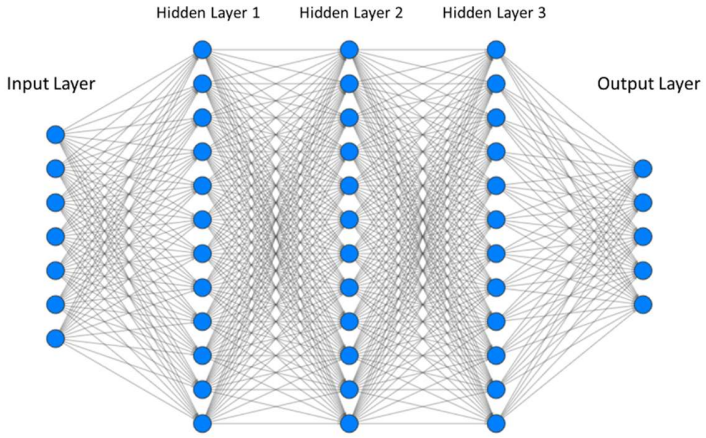


Figure 3. Deep Neural Network (Multilayer Perceptron).

Three families of Neural Networks can be distinguished nowadays: Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) [4]. These families have their own characteristics and used in variety of applications as for image classification, computer vision, speech predictors, etc. Although the three of them can be used for regression

analysis in some way or another, the first one is the most commonly used for this purpose. Then, given the characteristic of the problem considered, this is the kind of Network that will be implemented in the present work.

A Multilayer Perceptron Network is described by a stack of fully connected layers, which correspond with the input and output layer, and one or more hidden layers in between (see Figure 3). Considering the neural network as a function $y = f(x_1, x_2, \dots)$, the nodes of the input layer can be seen as the independent variables $x_1, x_2, \dots$, commonly known as features. The output layer will correspond with the y value (or values) that comes out when applying the function over the input nodes. In practice, the input and output layer are defined by two sets of data: the input and the output set. Finally, the hidden nodes in a network basically correspond with the function f that maps the relationship between the inputs and the outputs.

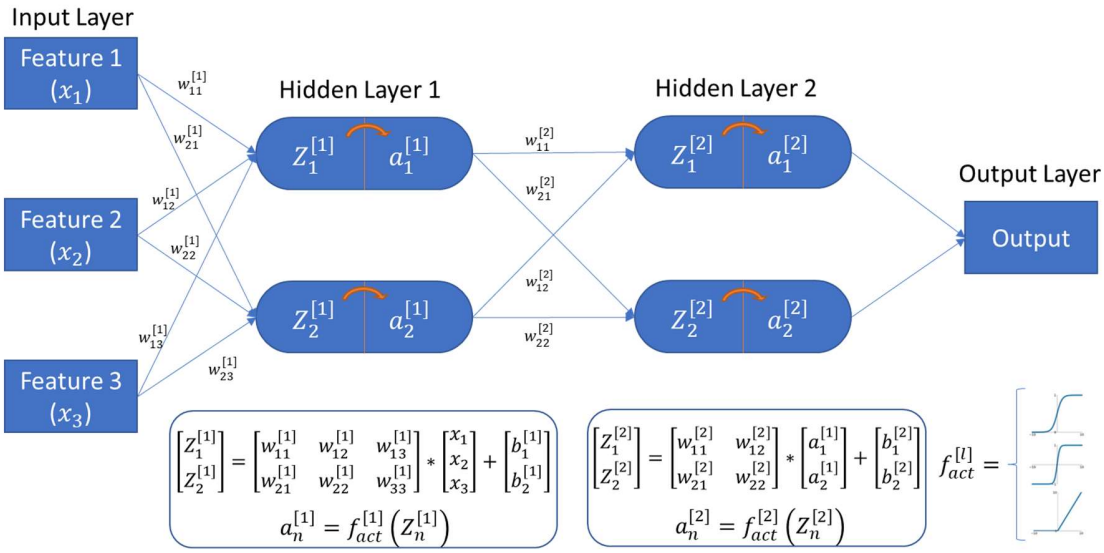


Figure 4. Hidden nodes and layers description.

The hidden nodes are mathematically defined as linear transformations over special non-linear functions that are applied on the results of the previous layers (see Figure 4). Their work is a key part in the regression analysis performed over the input and output set to find the unknown relationship between them. In most cases, this unknown relationship is a highly non-linear function that requires the use of several transformations, and an iterative "training" process, to find its intricated form. In general, it could be said for a Neural Net, that the more nodes and layers it has, the more able the network is to capture the non-linear patterns within the training set.

As an overview, the training process consists of the continuous updating of the parameters that affect the linear summation of the inputs to a hidden node. Those parameters are known as the weights and biases (w and b in Figure 4). The training is performed in two ways: in parallel over the nodes in a layer, and in series across the layers with a certain procedure. This procedure comprises 3 steps: Forward propagation, backward propagation and parameters updating. The first step applies the linear transformations over the input features layer by layer until the output layer is reached. The second step starts by calculating the difference between the predicted results of the net and the expected ones (which is the output training set) through a so called "Loss function".

Then it takes the information collected and goes all the way back across the Net until the first layer by calculating the derivatives of the different transformations applied on each layer. The last step updates, e.g. by a gradient descent method, the original weights and biases used in the first step with its respective derivatives found in the second step. This procedure iterates enough times to make the loss function decrease and converge to a certain value as close to zero as possible.

2.2.2. Main Hyperparameters

Every Neural Network is built over a set of hyperparameters[1] that controls its size, non-linearity behavior and predicted results. In an MLP Network there is a significant number of hyperparameters that play a role in the model selection [5]. However, only the most relevant were considered in the present work and are presented in the next list:

1) Hidden Layers: Number of layers of the network between the input and output layer
2) Hidden Units: Number of nodes (neurons) of each layer
3) Loss Function: Calculates the difference between the predicted and the expected values at the end of the forward propagation step. This works as the "objective function" to be minimized.
4) Activation Function: Works as a mathematical "gate" between layers. This function is applied over the outputs of the nodes of a layer. The result is used in the linear transformation performed with the weights and biases of the next layer (see Figure 4).
5) Learning Rate: Scalar $\in (0,1)$ that multiplies the parameter's gradient from the backpropagation step. Controls each iteration's step-size along the gradient direction towards the Loss function minima.
6) Optimizer: Gradient descent method
7) Mini-Batch Size: Number of samples introduced in the network at a time to estimate the gradient. A training set is divided in a certain number of Minibatches.
8) Number of Epochs: Number of times all the minibatches (i.e. the whole training set) are introduced in the Network.
9) Weight Decay (Lambda): Regularization hyperparameter to reduce overfitting (Section 2.2.3).

The set of hyperparameters require a tuning process in order to find the optimum configuration to get the best performance from the neural network. This task, known as hyperparameters optimization, can be performed either in a manual way or by applying special algorithms as Bayesian Optimization, Random search or Grid searching methods [6], [7]. In general, the application of algorithms as grid searching is the preferred way for some applications [6], since they can come up with a more optimum set of hyperparameters than the manual way. However, when the model is large enough (with many hyperparameters) and/or a considerable training data is involved, the application of those algorithms will be limited by the available time and computational resources.

[1] The word "hyper" is used throughout this report in different contexts. When mentioned in the term "hyperparameters", it refers to those parameters which define the neural net structure, architecture and function such as number of layers, nodes, type of activation function, etc. On the other hand, when mentioned in dimensional context, "hyper" is a short word used to describe figures or spaces in more than 3 dimensions.

Therefore, the manual searching becomes in a useful and straight forward way to find, at least in a first approximation, the set of hyperparameters to be used in the neural network.

One important hyperparameter is the metric to be considered. This is a function that works as a tool to monitor and measure the network performance and allows to determine when it is ok to stop the training process. The metric helps also to determine which is the best model when choosing between different Neural Network architectures. In regression analysis, unlike classification problems, the metric should be able to work in a continuum regime. This means that the evaluated values are not binary, as in classification, but scalars that represent variables in the real domain ($\in \mathbb{R}$). Therefore, the comparison between the predicted and the expected values in a metric sometimes is taken as the same loss function used in the training, or just the percentual difference between the sets. In general, a good metric should be able to give information about the Neural Network in terms of robustness, interpretability and accuracy [8]. Thus, the task to define which metric to use in a Neural Network implementation project is not only very important but sometimes non-trivial at all.

2.2.3. Bias and Variance

As described in Section 2.2.1, an MLP Neural Network model must go through an iterative training process in order to estimate the function that maps the inputs and outputs layers. This process is performed upon a set of data pairs (inputs and outputs) known as the training set. However, during training, the network must be evaluated upon an additional set to test the effectiveness of the model predictions. This set, known as "cross validation", has the aim to validate the Neural Network model against data that never has been "seen" during the training process. This cross-validation strategy is also used in the hyperparameters tuning process. In practice, the cross-validation set could be randomly extracted from the data destined to create the Neural Network. Normally, the size of the cross-validation set is taken to be about 10% (or less) of the training set but it depends on the total amount of data available. It is also valid to separately generate an exclusive set of data for this end. Care should be taken so that the new data is valid in terms of variable range and distribution. Nonetheless, this often requires that part of the new set be added to the training set to avoid a large error in the cross-validation predictions.

When the loss function of the network model is evaluated upon the train and cross-validation sets at each epoch, two curves are generated. They describe the evolution of the error as the network goes through its training process. From them, it is possible to extract information of two important concepts used in Machine Learning which are: the bias and variance.

From technical definitions [9], the Bias "are the simplifying assumptions made by a model to make the target function easier to learn". In practice, it refers to the flexibility of a model to fit the training set. Low Bias means a high flexibility of the model; therefore, the predictions are well adjusted to the training set. High Bias means in contrast a lack of flexibility, so the model predictions do not fit well the training set. On the other hand, the Variance, "is the amount that the estimate of the target function will change if different training data was used". It refers to the model flexibility to changes in the evaluated set, or in other words, it gives account of the model robustness. Low variance means that the predictions of the neural network are similar regardless whether the network is

evaluated with the training or the cross-validation set. The contrary happens when there is high variance, large difference between model predictions when both sets are evaluated (Figure 5).

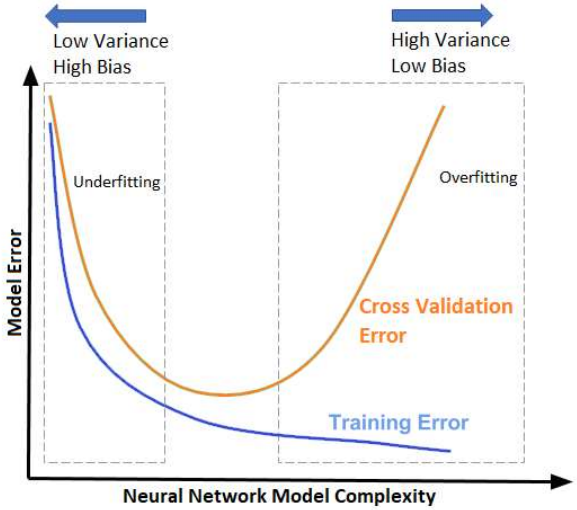


Figure 5. Bias and Variance Concept.

It is possible to have in a model low Bias and high variance or vice versa. When the first situation happens, it is said that the model is "overfitted". On the contrary, when there is high Bias and low Variance, the model is defined as "underfitted". Therefore, there is a trade-off in the bias and variance that allows the model to have the most suitable fit for the data that is used. The Figure 6 summarizes these "fittingness" concepts graphically.

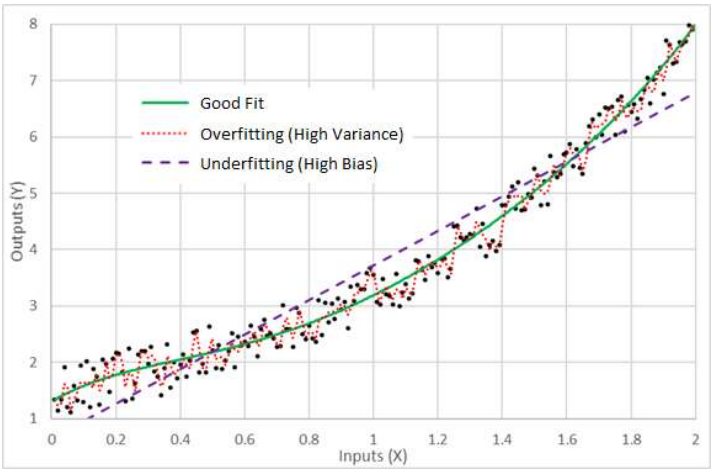


Figure 6. States of a model fittingness.

Once the hyperparameters of a model has been tuned up to have the best fit closeness-like state, sometimes a third data set, known as the "test set", will be defined. Although the cross-validation set can be extracted from a different distribution than the training set, the test set must be extracted from the same distribution as the cross-validation set. As the name suggests, this last set will be used to test the model against another set of "unseen" data to finally check once more the performance of the final network.

2.2.4. Frameworks

The very nature of the Neural Networks is perfectly suitable for any programming language. However, its implementation requires the usage of many functions and subroutines to get the desired result. Nowadays, Python is the most used programming language for this application and several libraries have been developed to facilitate this task.

Companies as Google, and some researching groups, have created very useful tools compiling several machine learning libraries into the so called “Frameworks”, making the process of creating a Neural Network very user friendly and straightforward. The list of available frameworks is large but some of the most known are: Tensorflow, Pytorch, Theano and Keras [10], [11]. The present work was conducted using Tensorflow 2.0 with a Keras layer on top.

3. METHODOLOGY

3.1. Methodology Overview

The methodology consisted of five stages:

1. **Plant model implementation and variables selection (Section 3.2)**. Definition of the full IGCC Aspen model and the variables to be collected for the Neural Network implementation.
2. **Mini model study (Section 3.3).** Definition of a Mini model from one of the full IGCC Aspen model blocks. This stage settled down the methods and codes for the data generation and collection. A preliminary Neural Network was implemented upon the Mini model to evaluate the effect of some hyperparameters on the Network performance.
3. **Data acquisition plan for full IGCC model (Section 3.4).** Based on the Mini Model outcomes, this stage defined the data acquisition plan for the full IGCC model.
4. **Neural Network Implementation (Section 3.5).** Setting up the Neural Network for the full IGCC model and main hyperparameters definition.
5. **Results Evaluation (Section 3.6).** The outcomes from the Mini model and full IGCC Neural Networks are described. Also, the results of a quick optimization study made upon the final Neural Network are presented

Figure 7, shows the overview of the five methodology stages. The detailed description of each of the stages is presented in the following sections.

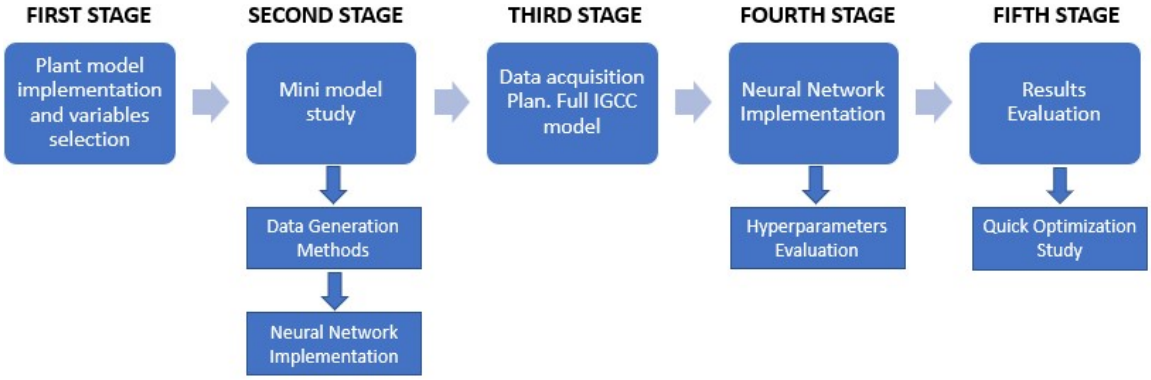


Figure 7. Methodology Overview.

3.2. First Stage: Plant Model Implementation and Variables Selection

This first stage consisted in the setting up of the plant model which defines the unit operations and processes involved in the IGCC plant. In Section 2.1 it was seen that an IGCC plant is a considerably large system which consists of several blocks, see Figure 1, that starts from the preparation of the raw fuel materials until the generation of the three main sub-products: power, methane and ammonia. Each block in turn contains several unit operations, often with recirculating streams and highly non-linear characteristics such as distillation columns and scrubbers. Beside the three main products, there are two "sub-products" collected as Sulfur and CO_2 for storage or industrial purposes. The Aspen model incorporates many of the traditional tools and methods developed to

model different kinds of chemical transformation processes and will be the reference against which the Neural Network will be evaluated in this work.

Typical control parameters were determined with inputs from experienced chemical engineers at AFRY. A selection process was then performed considering the relevance of each variable within the process while keeping a constraint on the total number of variables given the available computational resources along with the time limit for this project. The original IGCC model was also adjusted in several ways to more properly mimic the behavior of a real plant in operation.

3.2.1. Plant Model

The software used to perform the modelling and simulations was Aspen Plus Version 11. The modified main flowsheet that was used is presented in Figure 8. This model is derived from a template originally created by Aspentech and delivered as part of their Aspen Plus software Package. From now on this model will be referred to as the "full IGCC model".

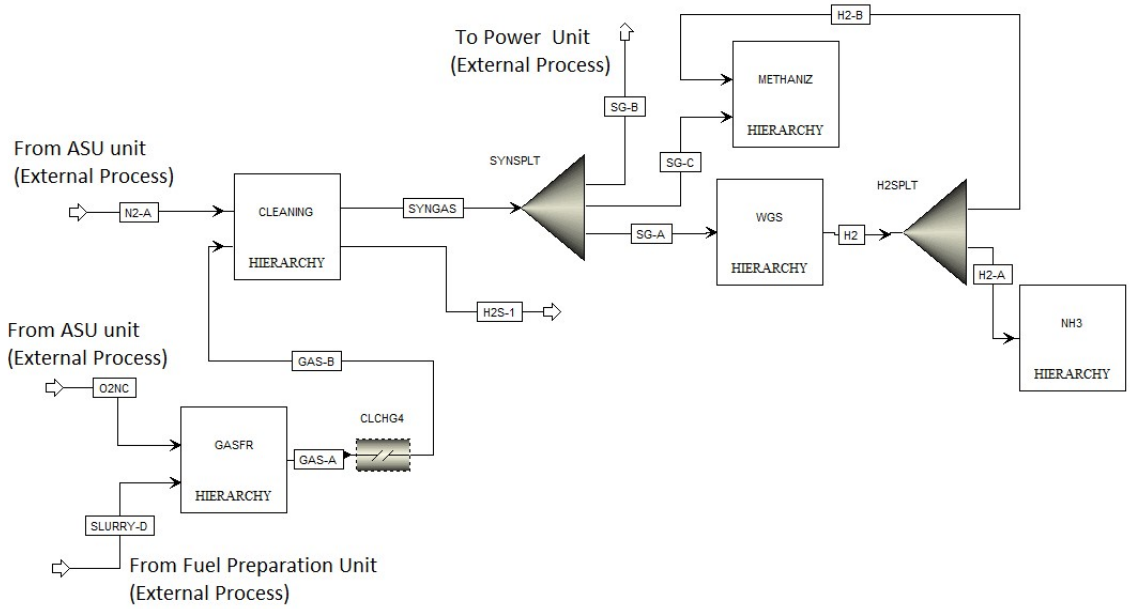


Figure 8. Main Flowsheet (see also Appendix A, Global Flowsheet)

The sub-flowsheets for each hierarchy block depicted in Figure 8 are shown in Appendix A. The whole IGCC plant model, as depicted in Figure 1, was deemed not suitable for this project as some blocks were set up in such a way that they would not properly respond to changes in input conditions. Some examples could be 'fixed conversion ratio' in a reactor, pumps or compressors with 'fixed efficiency' over an infinite flow range, or heat exchangers that perform both thermal and mechanical work on a fluid. Some of these limitations/restrictions were solved by modifying the model, but in some cases the time required was deemed not worth the effort. To simplify the model, the air separation unit, fuel preparation unit and the power plant unit were all removed. Instead these operations are assumed to be performed externally, i.e. the fuel is purchased pretreated, or the oxygen is produced by another company close-by etc. The properties of these External Processes are described in the next section.

3.2.2. External Processes

The external processes consist of the following blocks:

- Air Separation Unit
- Fuel preparation Unit
- Power Plant Unit

Each of these blocks have specific output streams whose properties were kept constant to reduce the degrees of freedom of the simulation.

- **Air Separation Unit**

Responsible for the separation of Oxygen and Nitrogen from the incoming air. Its products are Oxygen and Nitrogen with the mass fractions presented in Table 1.

Table 1. Mass fractions percentage of the O_2 and N_2 output streams from the Air Separation Unit.

	STREAM	
Element	**OXYGEN**	**NITROGEN**
N2 (%)	1.38	98.52
O2 (%)	94.39	1.11
AR (%)	4.23	0.37

- **Fuel Preparation Unit:**

The IGCC plant model incorporates a cofiring between Biomass and Coal. The Fuel Preparation unit crushes the Biomass and Coal and mixes it with water to create a slurry that feeds the gasifier. The proximate, ultimate and sulphate analysis of the fuel materials, named as PROXANAL, ULTANAL and SULFANAL respectively, are presented in Table 2. These values are the default ones assigned in the original IGCC Aspentech model.

Table 2. Proximate, Ultimate and Sulphate analysis of the Fuel Materials (the values are % on a dry basis except for the moisture)

		BIOMASS	**COAL**
PROXANAL	Moisture	5.7	9.54
	FC	28.13	50.90
	VM	67.06	39.45
	Ash	4.81	9.65
SULFANAL	Pyritic	0	2.44
	Sulfate	0	0
	Organic	1.24	0
ULTANAL	Ash	4.81	9.65
	Carbon	84.24	74.45
	Hydrogen	7.13	4.96
	Nitrogen	0.24	1.59
	Chlorine	0.15	0.07
	Sulfur	1.24	2.44
	Oxygen	2.19	6.84

The particle size distribution of the outcoming stream is depicted in Figure 9.

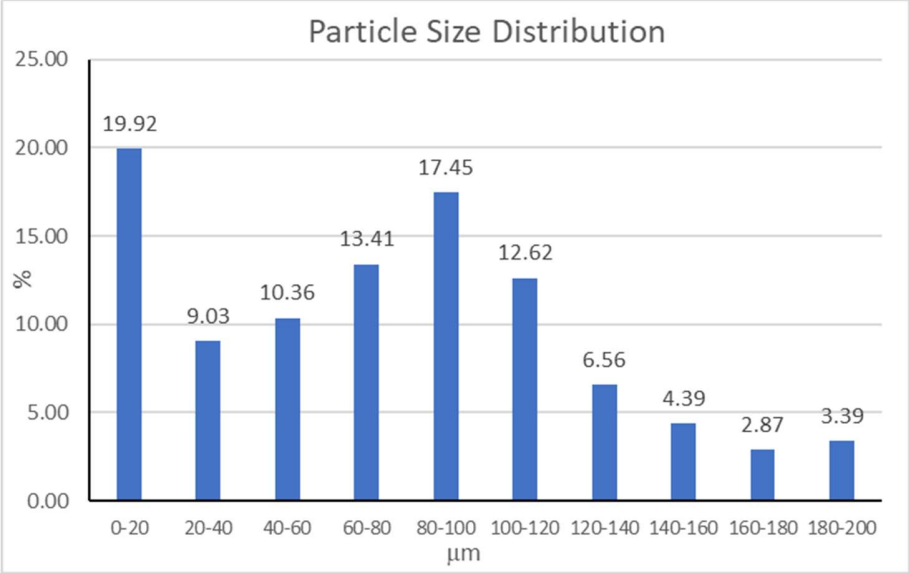


Figure 9. Particle size distribution of the output fuel mix

- **Power Plant Unit**

This unit receive a share of the syngas produced in the main plant and transforms it into electric power by means of a combined cycle system of gas turbines and a Heat Recovery Steam Generator (HSRG) unit. This plant is assumed as an external process with a constant combined cycle efficiency of 55%, which is the average value obtained for this kind of plants. The work generated will depend solely on the amount of incoming syngas and its H_2/CO ratio. Due to the complexity of the Power Plant Unit, and some modelling assumptions present within it, the accuracy of the simulations are not expected to be much affected by these simplifications but the simulation time and stability of the full IGCC model it was likely to have improved much.

3.2.3. Selection of the Input Variables

In order to generate the input layer for the neural network, it was necessary to select a set of variables that have an effect on the behavior of the plant and, at the same time, represent controllable parameters in a physical plant. Candidates for this set are: feed stream temperatures, mass flows, compositions, distillation column top temperatures, reflux ratios, reactor temperatures, etc. Once the variables were chosen, each one's value was picked randomly from an operational range previously determined (Section 3.4). A collection of values for the whole set of variables is referred to as the "sample set". This sample set was the precursor to generate the input layer that feeds the Neural Network (see Sections 3.3.5 and 3.4). A collection of more than one sample set from now on will be referred to as the 'input space'.

The number of variables chosen determine the degrees of freedom of the system. The more degrees of freedom, the more time and computational resources are needed to generate a representative input space to the Neural Network. Indeed, this increase could be related with a geometrical law.

As a rough mathematical approximation, it could be said that if 10 variables were chosen and each variable are varied over 7 values, the amount of sample sets needed to create a homogenous input space will be $7^{10} \approx 282.5$ million. This huge number is virtually impossible to handle with the available resources. Therefore, strategies as random space generators will be used to reduce the amount of data required to characterize the space (see Section 3.3.2). Nonetheless, the number of variables will still play a vital role in the data collection stage.

Hence, a careful consideration should be given in order to select the most relevant variables of the process. Given the blocks and processes present in the considered case, a list was created with the variables that fulfill the characteristics previously described (the list can be found in Appendix B). Out of the 22 possible candidates, 13 were selected heuristically based on chemical engineering experience. The variables are shown in Table 3, and its description is presented in Appendix B.

Table 3. Input Variables Selected

Block	Stream/Component	Variable	Number
GASIFIER	SLURRY-I	Biomass/coal mass flow ratio	1
	O2NC	Total stream mass flow	2
CLEANING	LEAN-1	Total stream mass flow	4
	N2-A	Total stream mass flow	5
WGS	LEAN	Total stream mass flow	9
NH3	Splitter PRIMSPLT	Split ratio for COOLG stream (To HX)	13
	Splitter SECSPLT	Split ratio for FD1-A stream (to BED1)	14
	Splitter RCYCSPLT	Split ratio for RECYC-4 stream (to recycle)	15
	Reactor BED1	Operating temperature	16
	Reactor BED2	Operating temperature	17
	Reactor BED3	Operating temperature	18
GLOBAL (Fig. A5)	Splitter SYNSPLIT	Split ratio for SG-A stream (to WGS)	21
		Split ratio for SG-C stream (to METHANIZ)	22

The streams and component names shown in column 2 can be found in the respective flowsheets in Appendix A. The column titled “Number” corresponds to the variable number in the list given in Appendix B. It is important to say that, due to the need to keep the number of selected variables as small as possible, there is a chance that some variables that were not included have similar relevance in a real IGCC plant as the ones selected. Thus, a broader study in the future could be done regarding this to include those non-selected variables into the input space.

3.2.4. Additional Considerations

The non-selected variables as well as different other parameters that were present in the unit operations of the model were kept fixed with their default values. Those default values correspond to the ones that were assigned in the original IGCC Aspen Plus model. The heat exchanger units present in the different blocks, were originally set up with fixed outlet temperature. These were then adjusted to fixed area to better mimic a real plant (different for each HX). The areas selected for each heat exchanger corresponded to the area at the program’s initial operating point. The heat

transfer coefficient U was calculated with a power law for flow rate taking as references the default flow values of the cold and hot streams.

3.3. Second Stage: Mini Model Study

In order to gain experience in handling the Aspen model and the Neural Network implementation, a preliminary model was set up. This model, from now on known as the "Mini model", was used to identify, although on a lower scale, the different challenges likely to appear in the full IGCC model. The mini model served also as a basis to develop the data generation scheme and methods for automatized the data collection process from the Aspen model. Additionally, the implementation of the hyperparameters suitable to train the Neural Network was evaluated on this step along with an estimation of the required number of samples, at least to an order of magnitude, for the task at hand.

The mini model was selected as one of the different subsystems of the process, i.e., the different blocks of the main plant. The selected block was the Water Gas Shift (WGS) and its detached flowsheet is presented in Figure 10. The selection was motivated by the non-linear behavior of the absorber and the recirculation streams. As highly non-linear models generally are more challenging, the WGS block was considered a good candidate.

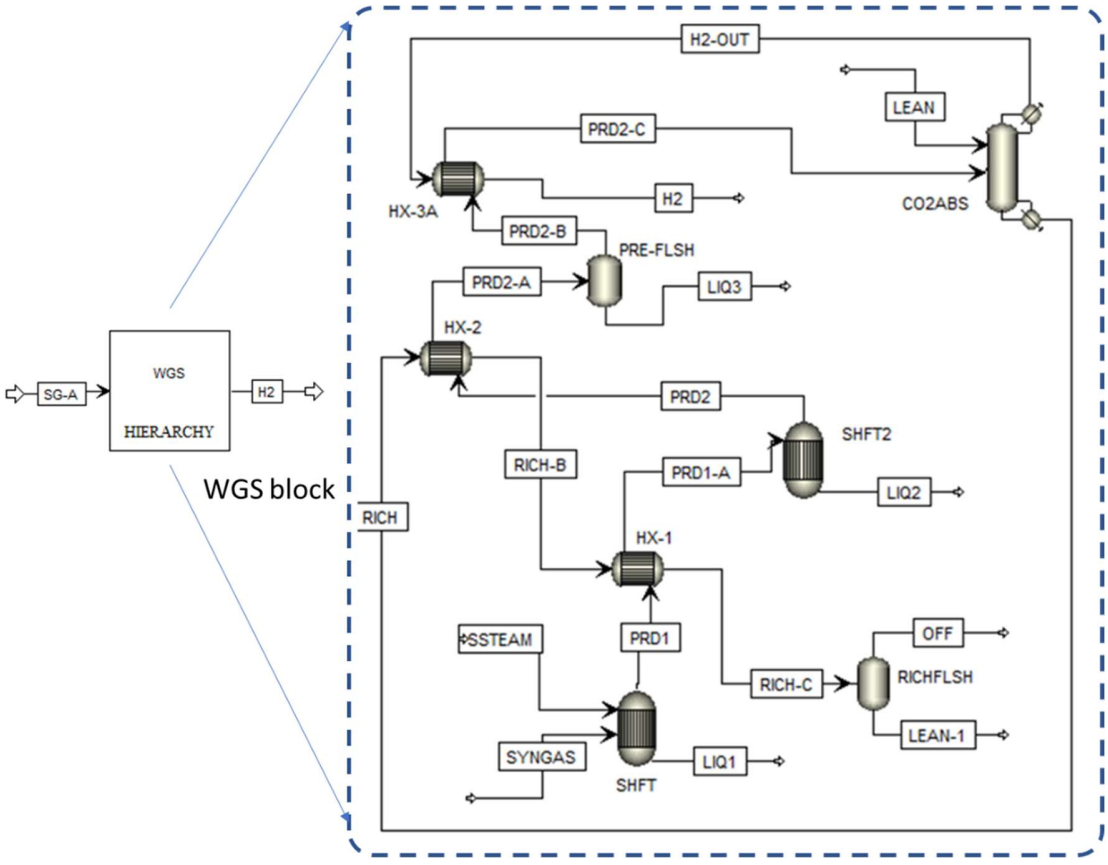


Figure 10. Mini model flowsheet

3.3.1. Variables Selection

Bearing in mind that this block was detached from the main process, some specific stream components had to be included in the inputs even though they would never be varied independently in a real plant. Thus, in this case the variables selected are presented in Table 4.

Table 4. Quick and Dirty model variables

				Op. Range	
Stream	**Number(n)**	**Variable**	**Default**	**OR_{Min}**	**OR_{Max}**
SG-A	1	Temperature (°C)	93.33	0.5	1.75
	2	Pressure (bar)	27.579	0.5	1.75
	3	Mole flow CO (kmol/hr)	767.77	0.8	1.75
	4	Mole flow H2 (kmol/hr)	544.491	0.7	1.25
	5	Mole flow of N2 (kmol/hr)	506.464	0.4	1.75
LEAN	6	Total Mole Flow (kmol/hr)	3923.574	0.4	1.05
	7	Temperature (°C)	- 40	0.8	1

The last 2 columns of Table 4 shows the limits of the operative ranges in relation to the default value (column 4). The ranges were obtained after performing a two-step dimensional analysis over the chosen variables. The first step corresponded with a unidimensional analysis to find the individual operative range for each variable. It started by assigning a large range between the minimum and maximum values (e.g. OR_{min}= 0.1 and OR_{max}=3), and then creating a set of 200 numbers distributed between 0 and 1. Those numbers will multiply the default value following the equation (2):

$$V_n = V_{0,n} * [OR_{min} + r * (OR_{max} - OR_{min})] \qquad (2)$$

Where V_n represents the calculated value of the nth variable, $V_{0,n}$ corresponds with the values of the "Default" column, and r is a number between 0 and 1. The equation (2) was applied over a single variable, while the other ones remain at their default values. This created a set of 200 samples (each sample represents a set with the 7 variables) that was introduced one by one in the model to get the respective results. The individual operative range for the variable is defined as the range where the model does not present convergence errors. The second step consisted in to take 500 "r-numbers" and apply equation 2 over a pair of variables with their respective individual operative ranges obtained previously (the rest of the variables remain constant). This way a bidimensional analysis was performed to check for regions where there were still convergence errors, and to select new more suitable variable ranges; in order to reduce the total number of errors when the automated data generation be launched later. The execution of these steps brought significant results, because the number of errors was significantly reduced from ∼35% to ∼1% with respect to the total amount of evaluated samples (check results in Appendix D). This translates in practice in a reduction of the total simulation time to get useful data to train the network.

Worth to mention, though, is that by shrinking the variable ranges, the entire input space, and thus the output space, that will be explored shrinks as well. Therefore, a completely error free data

generation process might not be desired if it requires the volume of the output space to be very small; restricting the use of the neural network to only a small region. For instance, if one variable range is reduced by 20%, the entire input space is also reduced 20%. Given a 7 variable input space, a 20% reduction in each variable reduces the (7 dimensional) volume of the input space by almost 80% $[1-(1-0.2)^7 \approx 80\%]$. Hence, there was a trade off between the total errors and the operative ranges, so that the space defined be wider enough to get a significant number of useful data in the available time for this step.

3.3.2. Random Space Generators

In order to create the input space, a collection of samples must be generated by applying equation 2 over a set of r numbers. Thus, this set of numbers must be able to create an input space that is representative enough to cover the whole volume enclosed by the operative ranges of the chosen variables, or in other words, the hyperspace of the degrees of freedom of the system.

As discussed in section 3.2.3, creating a homogeneous input space is unfeasible in practice given that each new sample set is generated by merely changing one variable at a time. Also, that way results unpractical. If the influence of some input variables over the outputs is higher than others, the "less important" variables might have required only a few datapoints while the "more important" ones would have to be sampled more densely. Therefore, the most appropriate approach to be followed is to cleverly change all variables in each new data set simultaneously, so the issue of having "less" or "more" important variables be not relevant anymore. In that sense, the methodology followed consisted in the use of a random number generator to fill up the whole hyperspace as homogeneously as possible for all its dimensions. By doing so, it is expected to create a distribution of data that be a good representation, in statistical terms, of the whole volume considered (see reference [12]). In this approach, If the number of datapoints within the input space is increased by a factor of 2, the total investigated space does not increase at all but the density of explored points within the space increases by a factor of 2.

Three types of random generators were evaluated:

- Montecarlo Random Sampling (MCS)
- Latin Hypercube Sampling (LHS)
- Quasi-Montecarlo Sequence with Sobol Numbers (QMCS)

Each of these sampling generators have special characteristics and ways to fill multivariable spaces by creating numbers between 0 and 1. A short description will be given next, but a detailed explanation about the methods and equations can be found on its respective references. The space created with those generators, with as many dimensions as variables chosen, will from now on be referred to as the "random space".

- **Monte Carlo Random Sampling (MCS)** [13]**,** [14]

Monte Carlo is one of the more classical techniques for random sampling used to calculate many impossible analytical integrals at very low cost. In this study, multiples samples are randomly collected trying to define the operative space of the variables as homogenous as possible. Those samples are drawn from a probability distribution that could be normal, uniform, weibull, etc. [15] Based on the "law of large numbers" from statistics, the more random samples are collected, the

better described the space will be. In practice, the method was implemented using the numpy.random.rand function (uniform distribution) with a specific seed over the dimensions of the space considered (7 in this case).

- **Latin Hypercube Sampling (LHS)** [16]**,** [17]

The Latin Hypercube sampling method take the characteristics of Monte Carlo sampling in terms of its randomness but ensuring that the set of random samples is a good representative of a distribution [18]. Sometimes it can reduce drastically the number of samples required to define a system. In one dimension, it consists in dividing the cumulative distribution function into a number of equally distributed partitions and then choose a random point in each partition. In this case, the method was implemented on a multidimensional space by the means of a library called pyDOE with the function lhs.

- **Quasi-MonteCarlo Sequence with Sobol Numbers (QMCS)** [19]**,** [20]

This is a technique within the low-discrepancy sampling methods, which unlike the classical Monte Carlo, the samples are not necessarily randomly independent but are based on deterministic sub-random sequences. This is promised as a high convergence technique for integral applications [21]. A multivariate application of this technique has the goal to generate samples distributed in a way that approximates a uniform distribution covering the entire space. The method was implemented by a special function found in a library written in Python 2.3, that had to be extracted and manually modified to take correctly the Sobol sequences, and that be able to run on Python 3.7.

3.3.3. Input Space Sorting Methods

The input space, obtained by applying equation (2) over the entire random space generated, corresponds to a two-dimensional matrix with as many columns as variables being considered (7 in this case).

Two of the main concerns when introducing data into the Aspen model is to reduce the time per simulation and the total number of convergence errors for the whole input space. One of the common sources of an increased calculation time, and responsible perhaps of some errors also, is that the input data introduced into the model takes “big jumps” between different samples, i.e. there is a large change in the input conditions. A small change (or “small jump”) in the input variables means that the model will be close to convergence already from the start. The model must only make minor adjustments to converge and the resulting calculation time is thus short. On the other hand, a big change in all, or several input variables, requires the model to start far off from convergence and will need to “travel a long way” to find convergence again. This may result in longer calculation time and, potentially, more errors if the convergence path is highly non-linear. Hence, one important step before sending the list of inputs to the model, is to determine what sorting to be applied to the data. In practice, this order is given by the arrangement of rows in the random space matrix. Therefore, 3 different methods were tested, and its results presented in section x (Table Ap 1 in Appendix D).

- **Random Insertion**

No sorting method is applied here. Each sample set is fed into the simulator in the same order it came when the equation 2 was applied to the random space generator.

- **Hypercubes Sorting Samples (HCS)**

This method consists in dividing the multidimensional space into small subspaces and then sort them, so the distance between jumps can be reduced. This method comprises of two steps. First, small hypercubes are created from the input space by splitting each dimension in 2 or more parts (in the Mini model 7 variables were selected, therefore there are 7 dimensions). The combinatory of the one-dimensional sub-ranges obtained allows the creation of small regions in the multidimensional space that will be called "sub-hypercubes". In Figure 11, it is presented the cases for two and three dimensions. As seen there, each sub-hypercube will contain a number of samples that is directly related to the number of divisions made over each dimension and the sample distribution. In the second step, the matrix that contains each sub-hypercube is sorted from the right most column to the left most one. This way, the sub-hypercubes are sorted so that each consecutive sub-hypercube in the matrix represent an adjacent sub-hypercube of the whole hyperspace. Once the final step is done for all the sub-hypercubes, the input space will be ready to be introduced into the simulator.

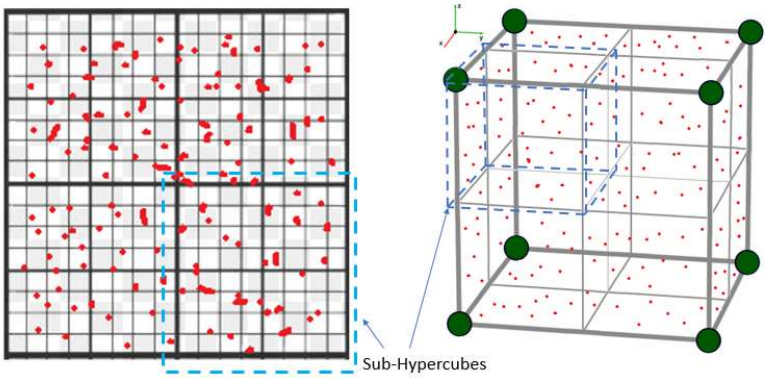


Figure 11. Sub-Hypercubes in the input space. 2D (Left) and 3D (Right) representation

As can been in Figure 11 (left), each sub-hypercube will contain a different number of samples because the datapoints are randomly distributed. If there are more splits over each axis, the number of samples will be reduced. Mathematically, the number of hypercubes is related as: $N_{hyperc} = (1 + N_{div})^{\#dim}$, where N_{div} is the number of divisions ($N_{div} = 1$ in the 3D case of Figure 11) and $\#dim$ are the dimensions (7 in this case). It could be the case that if the sub-hypercube is too small some of them will be empty. So, there is a tradeoff that will depend on the size of each sub-hypercube and the number of data points. By default, it was assigned a $N_{div} = 1$, so the number of sub-hypercubes created was 128 for this case.

- **Traveling Salesman problem modified for hypercubes (TSPH)**

This method is based on the Traveling Salesman problem which focuses in to look for the route that connects all the cities in a traveling plan in the smallest total traveling distance possible (Figure 12) [22]. This problem has been widely studied for the useful application in routes minimization problems, and one of the common algorithms used is known as 2-Opt Algorithm [23]. In this work, the 2-Opt Algorithm was adapted from reference [23] and applied on a multidimensional Euclidean metric with the possibility to assign weights on individual dimensions. Since the convergence of the

simulation model is not equally sensitive to all parameters, the weights can be used to prioritize the sorting so that the least sensitive parameters are allowed to make larger changes between runs. Therefore, but not strictly, the data will be sorted in order to decrease the total distance between datapoints according to the variables that are more sensitive, so that the total path is the most "convergence friendly path" for the simulation.

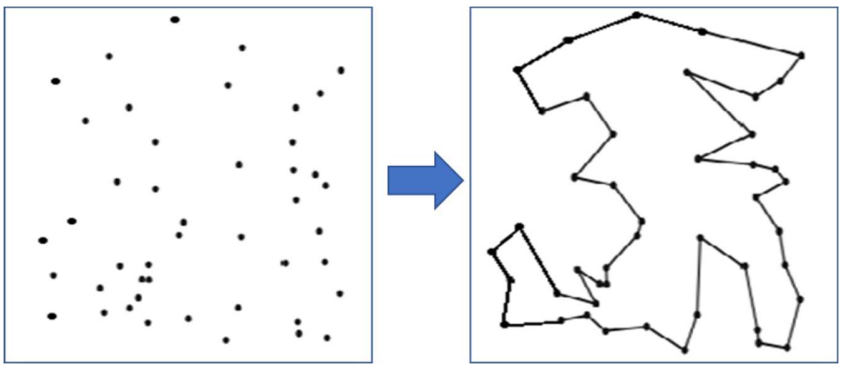


Figure 12. Traveling Salesman problem graphical description.

Hence, in practical terms, when the algorithm is applied over a sub-hypercube, it sorts all the samples contained within the region for the shortest total hyper-path that connects the points given the defined metric. This is done until a given convergence threshold is reached (0.01 as default). The procedure is then repeated for all sub-hypercubes in the given space.

After evaluating the different sorting methods in terms of number of convergence errors the best sorting was TSPH (see Table Ap 1 in Appendix D). Finally, based on the results comparison when evaluating the different random generators methods (see Table Ap 2 and Table Ap 3 Appendix D), the final method scheme to create the input space was: **QMCS + TSPH**.

3.3.4. Python Interface

Since the number of samples required to train the network is expected to be significant (in the order of thousands at least), the work of initializing the simulations of the whole input space had to be automatized. A Python interface was developed based on a state-of-the-art code [24] that allows to connect with the Aspen Plus COM interface. Once the connection is established, it is possible to control certain functions on the console with the COM interface (as reinitialization and simulation), and get access to the stream properties through the Nodes Tree in the Aspen variable explorer. The work of the interface is basically to a) create and sort the input space, b) introduce the different samples into the Aspen console, c) run the simulation one by one and d) collect the results obtained for each sample. Several error handling subroutines were also introduced to make a robust interface to be able to work for extensive periods of time with minimum supervision. The random generators were implemented with the help of special libraries modified for this project. The interface code was built on Python 3.7.

3.3.5. Neural Network Implementation. Mini Model Case

The input and output layers of the MLP network were defined with the properties of the input and output streams of the WGS block respectively. The input and output streams showed in Figure 10, are presented in Table 5 together with the properties that will be extracted for each one of them.

Table 5. Input, Output streams and properties to be read from the mini model.

Streams	
Input	**Output**
SYNGAS	H2
SSTEAM	OFF
LEAN	LEAN-1
	LIQ3

Properties (*)
Temperature, Pressure, Enthalpy, N2, O2, AR, H2O, CO, CO2, COS, H2S, H2, CH4, MeOH

(*) The mole flow was taken as the value to be read from the chemical components listed above.

The properties for each stream listed in Table 5, are collected when a sample set is introduced into the Aspen model and simulated. All the values are then stacked together in a column vector and this conform the input and output layer as it is shown in Figure 13.

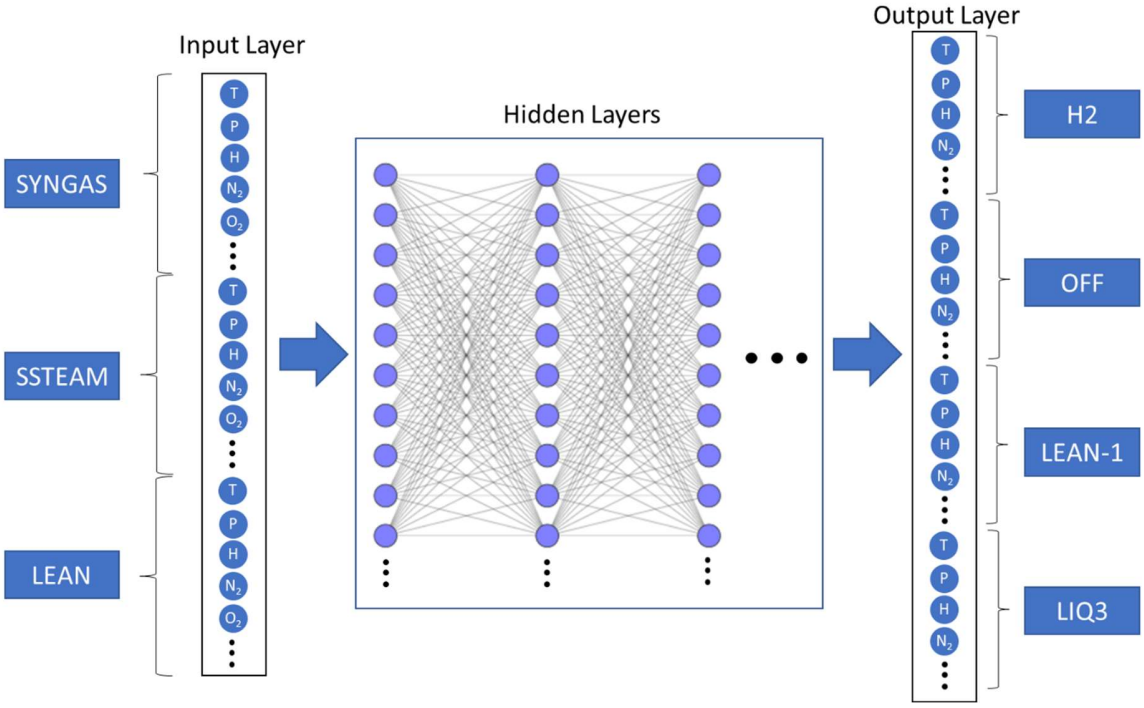


Figure 13. Mini model Neural Network Implementation

Once the input and output layer were established (see definitions on Section 2.2.1), the next step is to define the different hyperparameters to build up the Neural Network. For the mini model, there are 46 features in the input layer and 56 nodes in the output layer.

The Neural Network was developed using the subroutines of the Keras library which is based on the Tensorflow framework [25]. Due to the high level of operations that are iteratively performed in the

training process, the need for Graphic Process Units (GPU's) assistance is practically a must. Given the limited computational resources available for this project, after some research, it was found a very useful tool that offers significantly powerful hardware to perform this kind of calculations. Google Colaboratory, commonly known as Google Colab, is a project developed by Google that offers a free cloud service with access to special GPU and TPU (Tensor Process Unit) hardware to develop deep learning applications at a researching level. Its environment is based on Jupiter Notebooks [26] to write and execute code in Python. This service provides access to different types of GPU that vary over time. The GPU's available are Nvidia K80, T4, P4, P100 and V100. When a connection is established with the service, a virtual machine (VM) is loaded with 12GB of RAM and one of the GPU's. The VM can be used for machine learning applications, among others, during a period of 12 hours per session. A more detailed description about what Google Colab is and what are the different process units used for can be found in reference [27].

3.3.6. Neural Network Hyperparameters Study. Mini Model Case

As discussed in Section 2.2.2, there are several methods to perform the hyperparameters optimization. In this project, the method chosen was "Coordinate Descent", which is part of the manual methods set [28]. The procedure consists in systematically change one hyperparameter at a time while the other ones remain constant looking to reduce the validation error in the results (see section 2.2.3). The different hyperparameters used in the Mini model are described next:

1) **Number of Nodes and Layers:** The starting architecture was one hidden layer with 200 nodes. From this point different arranges were tested by increasing the number of layers and nodes to see the impact on the results. At the end, the Andrej Karpathy recommendation [29], [30] was followed: pick a large architecture to induce an overfitting state, and then apply regularization techniques to reduce the variance. This method also allowed to create a clearer picture about the effect on the train and cross validation curves when other hyperparameters are varying (see section 4.1.1).
2) **Loss function:** Mean squared error (mse). (See Appendix C)
3) **Batch Size:** Starting size – 32 samples. Several batch sizes were considered to see the effect of this hyperparameter (see section 4.1.2).
4) **Number of Epochs:** Between 400 and 1200 epochs. This number changed depending on the architectures and the conditions to get the plateau state in the error curves.
5) **Optimizer and learning rate:** Adam (Learning rate=10^{-5}). Other optimizers were also studied (see section 4.1.2).
6) **Lambda Regularization:** An L2 regularization study was performed for different lambdas. (see section 4.1.2)
7) **Data Preprocessing:** As a way to reduce the relative numerical differences between variables, the default preprocessing transformation chosen in this case was the standardization (Stnd).

Given the nature of the input data, where the orders of magnitude could be very different between variables, the definition of the kind of metric to be used resulted challenging till some extent. Several tests were performed applying a metric defined as the relative percentage difference, but many errors and/or very high values rose when some variables were zero, or close to it, in the output data. Also, the loss function (i.e. mse function) and "mean absolute error" (mae) function were considered

as metric candidates because of their zero-tendency decaying behavior as the training evolves. However, the number given by those functions turned out to be a bit too abstract to interpret in practice. Making it very difficult to develop an insight about the performance of the network and its related accuracy.

When the above-mentioned metrics failed in describing the accuracy of the neural network, a vectorial concept over a multidimensional space was then creatively implemented. Considering the nodes of the output layer as forming a linearly independant vector basis in $\mathbb{R}^n$, where n correspond to the number of nodes, each output set can be seen as a hyper vector in a 56th dimensional Euclidean space. Therefore, the predicted and expected sets can be seen as hypervectors $\vec{Y}_{predicted}$ and $\vec{Y}_{expected}$ respectively that describe the next relation:

$$\vec{Y}_{predicted} = \vec{Y}_{expected} + \vec{Y}_{res} \qquad (3)$$

Where $\vec{Y}_{res}$ correspond to the resultant vector formed between them. This expression can be better described in a two-dimensional representation as presented in Figure 14.

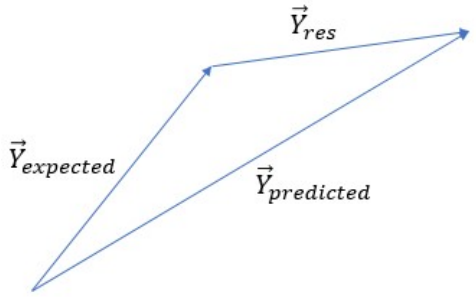


Figure 14. Two-dimensional representation of the vector concept applied over the output layer.

Rearranging equation (3):

$$\vec{Y}_{res} = \vec{Y}_{predicted} - \vec{Y}_{expected} \qquad (4)$$

$$\Rightarrow \left|\vec{Y}_{res} \cdot \vec{Y}_{res}\right| = \sum_{i} \left(\vec{Y}_{predicted} - \vec{Y}_{expected}\right)^2 \qquad (5)$$

The i term corresponds to the i^{th} component of the respective hypervector. The equation (5) defines the magnitude of the resultant vector, and can be seen as a measure of the Neural Network error. Based on this expression, the metric used was:

$$Metric = max\left\{\left|\vec{Y}_{res} \cdot \vec{Y}_{res}\right|\right\} \qquad (6)$$

In practical terms, equation (6) represents a function that will follow the maximum resultant vector from the whole set. In other words, it follows the output dataset with the maximum error between the predicted and the expected values. By implementing this vector as a loss function for the Neural Network its value at the end of the training would equal, not the average error over the entire data set, but the average error of the least accurate output variable. All the other predicted sets would

have a smaller error by consequence. The custom metric was implemented in the code by the means of a Python class callout applied in the ".fit" function of the Keras framework.

It is important to remark that at the end of the Mini model study, it was found that this was not a successful metric. Possibly because it continuously switched between different sets as training proceeded and, thus, obtained a somewhat erratic behavior. It also posed some problem in evaluating the neural network at all during the times where the vectorial metric was large. Therefore a new metric was defined for the full IGCC model which is described in Section 4.2.1.

3.4. Third Stage: Data Acquisition Full IGCC Model

This stage is the starting point of the study case and deals with the Data Acquisition of the full IGCC model. Based on the knowledge acquired from working with the Mini model, this stage started with a development plan to collect the data whose objective was to a) establish the path to follow in the data acquisition, and to b) determine how this process will be conducted for the available time and computational resources.

The workflow is shown in Figure 15.

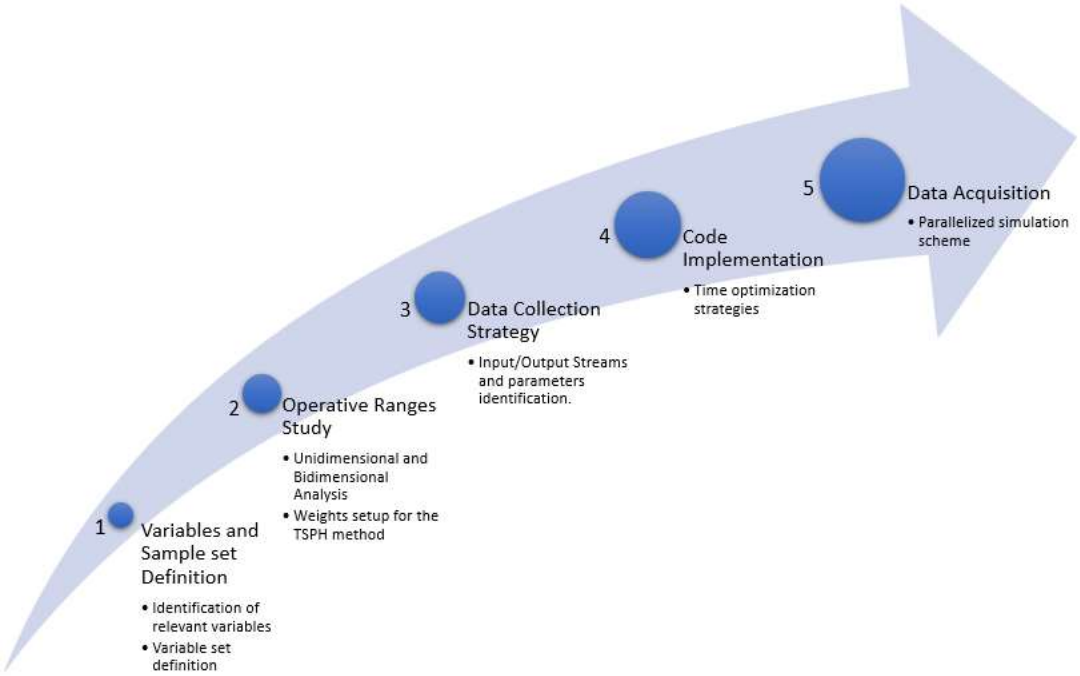


Figure 15. Data acquisition workflow for the full IGCC model.

The chart above presents the development plan in 5 steps. The first step, Variables and Sample set definition, correspond to the information presented in Section 3.2.3 regarding the selection of a suitable set of variables that feeds the simulation. Following the same procedure as of Section 3.3.1, the second and third steps are summarized in Table 6. The graphs obtained from the bidimensional analysis are shown in Appendix E. After this analysis was made, the number of convergence errors respect of the total amount of simulations was 40%.

Table 6. Set of Variables and Operative Ranges

				Op. Range			
n-th	Variable Name (V_n)	List	$V_{0,n}$	Max	Min	Weights	Description
1	Ratio_SLURRY_G	1	0.5	0	1.99	1	Ratio of biomass in the total slurry flow
2	Massfl_O2_G	2	3505.45	0.75	1.34	1	GASIFIER Oxygen Total massflow
3	Massfl_LEAN1_C	4	36296.23	0.8	2	1	CLEANING Lean1 stream total massflow
4	Massfl_N2_C	5	3401.94	0.6	2	0	CLEANING Nitrogen mass flow
5	Massfl_LEAN_W	9	3923.57	0.84	1.1	1	WGS Lean stream total massflow
6	Split_COOLG_N	13	0.861	0.4	1.1	1	NH3 Spliter
7	Split_BED1_N	14	0.2	0.4	4	0	NH3 Spliter
8	Split_RECYC_N	15	0.1	0.5	2.5	0	NH3 Spliter
9	T_BED1_N	16	400	1	1.5	1	NH3 Temperature Bed1 reactor
10	T_BED2_N	17	400	0.9	1.3	1	NH3 Temperature Bed2 reactor
11	T_BED3_N	18	400	0.9	1.5	0	NH3 Temperature Bed3 reactor
12	Split_WGS_S	21	0.15	0.9	4	0	Global splitter
13	Split_METH_S	22	0.065	0.01	5.5	0	Global splitter

The name of the columns refers to the Table 3 discussion. The “weights” values are the ones used in the TSPH method (see Section 3.3.3).

The fourth step consisted in the code implementation. The Python interface created for the Mini model was modified in order to handle the larger number of variables to be collected from the blocks inside the full IGCC model. Some new strategies were also implemented to further reduce the data collection time per simulation, for instance, the optimization and re-localization of some time-consuming subroutines were enhanced. The most important strategy for minimizing the data collection time was the identification of the relevant stream attributes. Table Ap 5 and Table Ap 6. in Appendix E, show the different attributes chosen for each stream. In total 147 and 335 stream attributes were read for the input and output set, respectively. After implementing these strategies, the data collection time was reduced from ∼60 seconds to ∼6 seconds per simulation.

The development plan ended with the data acquisition step. The total time spent to get 5000 useful data set (discounting the datasets with errors) was about 50 hours on a computer with an Intel core i5-8500 CPU at 3.0 GHz. Since the amount of required data was expected to be more than 5000, given the time limitation, a parallelized simulation scheme was therefore created in order to divide the simulation load over 3 computers with similar performance characteristics. The working scheme consisted of creating a large random space in 13 dimensions with the QMCS generator (200000 datasets) and split it in several batches of 5000 datasets. The specific random generator was chosen based on the results presented in Appendix D (see Table Ap 2 and Table Ap 3). After performing the TSPH sorting method over each batch (best method according with Table Ap 1, Appendix D results), each computer was assigned a set of 2 batches to simulate. Once a computer had finished, another 2 batches were loaded until a significant number of useful simulations (∼52000) had been performed. At the end, all the results were collected and manually stacked together to create the file for the Neural Network implementation.

3.5. Fourth Stage: Neural Network Implementation Full IGCC model

The job here focused on the creation of the Deep Neural Network for the full IGCC model. The knowledge acquired in the Mini model study also served as a starting point for the network implementation. Based on a coordinate descent method, the hyperparameters optimization was done following the sequence depicted in Figure 16.

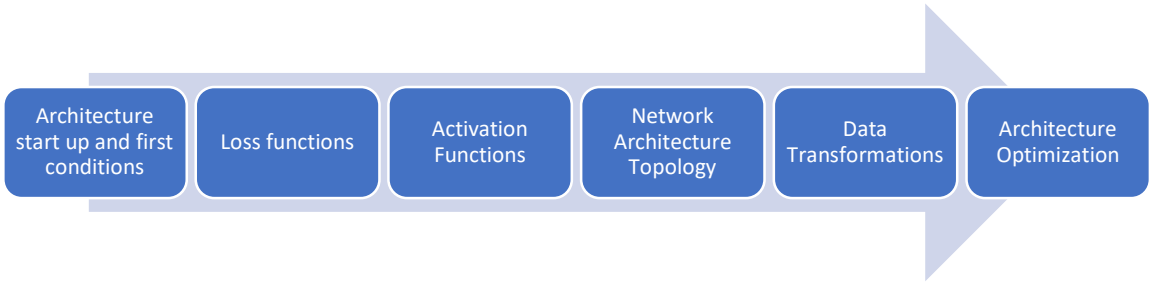


Figure 16. Process flow for the Neural Network implementation of the full IGCC model.

As shown in the process flow of Figure 16, the Neural Network construction was performed in a step-sequenced way. This process structure grounds its concept in what is known as the "Pandas" approach. There are two ways to build up a Neural Network model: The Pandas and Caviar approach. Both are based on a naturalistic concept, and the selection of one or the other depends on the available infrastructure to construct the required network [31], [32]. The "Pandas" approach is named after how that Panda bears raise their children. Here, all the "care and nurturing" is focused on one model at a time. Therefore, this approach is suitable for projects where the computational and/or workforce resources are limited. On the other hand, the "Caviar" approach resembles that of fish reproduction: Laying down dozens or hundreds of models at a time and then pick up the one with the best curve. This approach is chosen when there is enough infrastructure and resources to build parallel schemes.

The different set of hyperparameters tested in each step are shown in the list below. The detailed description of the Loss functions, Activation functions and Data transformations can be found in Appendix C.

1) **Loss functions:** *mse, mae, Huber, Logcosh, Vecloss.*
2) **Activation Functions:** *ReLU, leakyReLU, ELU, Swish.*
3) **Topology:** *Decremental Nodes, Incremental Nodes*
4) **Data Transformations:** *Standardization, MaxMin Scaler, Quantil Normal, Quantil Uniform*
5) **Architecture optimization:** Several architectures were tested to end up with the optimal one that will be used in the optimization study.

The results and analysis of the different methods applied are presented in Section 4.2.

3.6. Fifth Stage: Results Evaluation and Quick Optimization Study

The study concluded with the presentation of results from the full IGCC model. It discusses the different Neural Networks implemented and the final predictions against the Aspen model results. The optimal Network architecture obtained at the end of the process described in section 3.5 was used to make a test validation, and to perform a quick global optimization study to evaluate calculation efficiency of the network compared to that of the Aspen Plus model. Finally, a further discussion about the possibilities to apply the studied concept on real data, with all its limitations in matter of data acquisition and parameters evaluation, is also presented.

4. RESULTS AND DISCUSSION

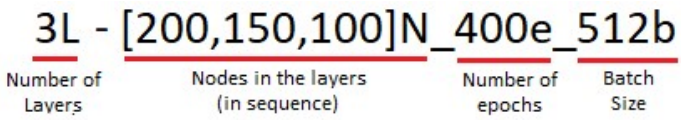
4.1. Mini Model Results

The Mini model study had the aim to gain knowledge about the MLP network implementation. Therefore, the process followed was oriented to have a clear view about the impact of the different hyperparameters upon the network (see section 3.3.5). This process consisted of the following steps:

1) Preliminary network architecture evaluation
2) Main Hyperparameters effect (Batch size, learning rate, Optimizers)
3) Definition of the optimal train set size

The results and analysis of each of these steps will be presented in the next sections. In Section 4.1.3 a summary with all the concepts and processes followed in this stage is presented.

The general conventions used to describe the models are:



4.1.1. Network Architecture Evaluation

Several network architectures were evaluated with the standard conditions presented in Table 7.

Table 7. Standard Hyperparameters for the Mini model neural network.

Loss Function	mse
Metric	Vecmax
Activation Function	ReLU
Optimizer	SGD
Learning Rate	1E-02
Minibatch Size	128
Data Preprocessing	Stnd
Train set size	20000
Cross validation Size	2000

Vecmax is the metric whose formulation is described in Section 3.3.6. The data preprocessing chosen was standardization (Stnd) [33]. The Train/Crossv sets split was performed with the "train_test_split" function included in the *sklearn* library for Python. The loss and metric values for the architectures tested at the end of the last epoch is presented in Table 8 and in Figure 17 the 1L-[100] and 3L-[200,150,100] cases are presented.

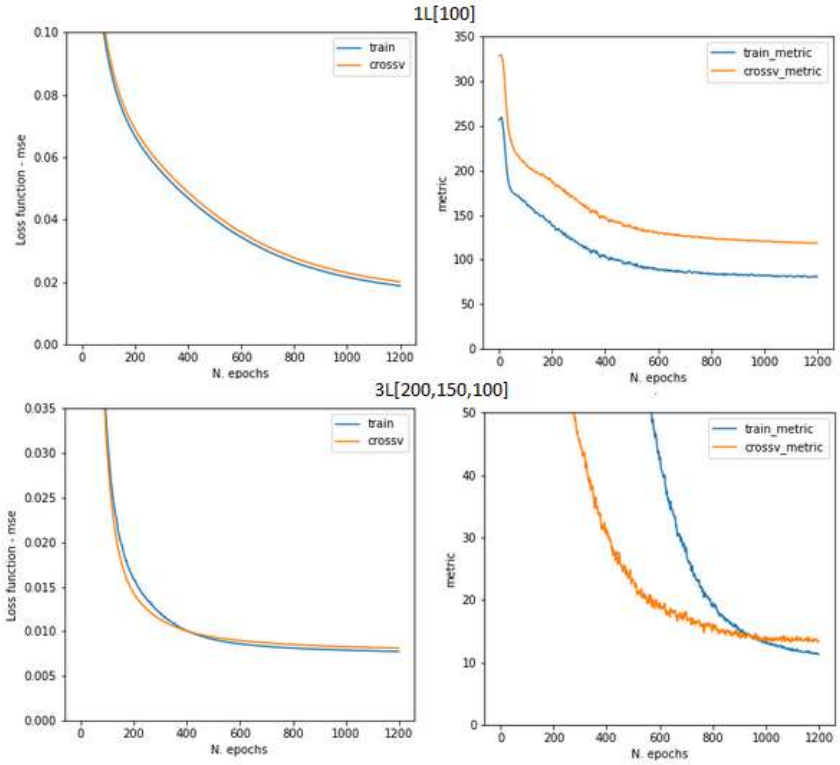


Figure 17. 1Layer and 3Layer model results.

Table 8. Loss function and Metric Results for the different NN architectures evaluated.

Model Architecture	LossF Train	LossF Crossv	Metric Train	Metric Crossv
1L[100]	0.0187	0.0200	91.3	128.0
1L[200]	0.0171	0.0185	86.0	132.4
2L[100,100]	0.0113	0.0122	60.4	100.6
3L[100,100,100]	0.0083	0.0087	15.6	24.2
3L[200,150,100]	0.0078	0.0083	11.2	13.1

As it can be seen in Figure 17, the loss function follows an exponentially decaying behavior which is characteristic of the training process. In all the cases, the training and cross-validation (crossv) curves are very close together, which means the model has low variance. The metric curve offers additional information about the error of the network, specifically the maximum error, and it decays too as the network is being trained.

Sometimes was found during this study, that the train curve was above the cross-validation curve. This counterintuitive behavior is probably because of the presence of outliers in the considered set. An outlier is defined as a sample that is statistically far away from the set distribution. Then, since the training set has the largest share of samples (10 times the crossv set in this case), not only the number of outliers will be in proportion greater than the crossv, but there will be a larger chance to find very separated outlier samples within the set. In view of the fact that the loss function is described as the average of the squared difference (see Appendix C), a large number of outliers in

the train set with a significant separation will have a direct impact on the set statistics that could lead to an averaged error higher than the crossv set. In simple words, the presence of outliers can make the crossv set simpler for the network to resolve than the train set. This effect can remain during several epochs until eventually the train set loss drops below the crossv loss (as can be seen in Figure 17 for the 3L case). This large-outliers hypothesis could also explain the similar behavior presented in the metric results. The nature of the outlier samples will be discussed in Section 4.2 (full IGCC results).

Another plausible cause is because the train loss is being measured during each epoch while the crossv loss is measured after each epoch. This last explanation, is strictly related to how the keras .fit function introduces the data into the network, but it was disregarded because the outliers-hypothesis seems to be the most probable reason given the results obtained in the different cases presented. For the sake of information, in reference [34] other possible causes can be found when this kind of results arise in machine learning problems, but most of them are not relevant for the present problem.

From the results of Table 8, it is possible to see that the more nodes and layers that are added to the network, the more reduction in the loss and metric error is obtained. There is actually not a consensus in the number of nodes or layers that should be used for a specific neural network application. However, to reduce the spent time trying to find a suitable number of layers and nodes, one of the most recommended ways is the overfitting induction (see reference [35], [36]). The method consists of implementing of a relatively deep neural architecture in order to induce an overfitted state, and then manipulate the other hyperparameters and regularization techniques in order to reduce it. Based on this method, a 6 layers network model was tested, and the results are presented in Figure 18. The model nodes are stacked as: 6L-[650,550,450,350,250,150]N. From now on, this model will be known as the 6L model.

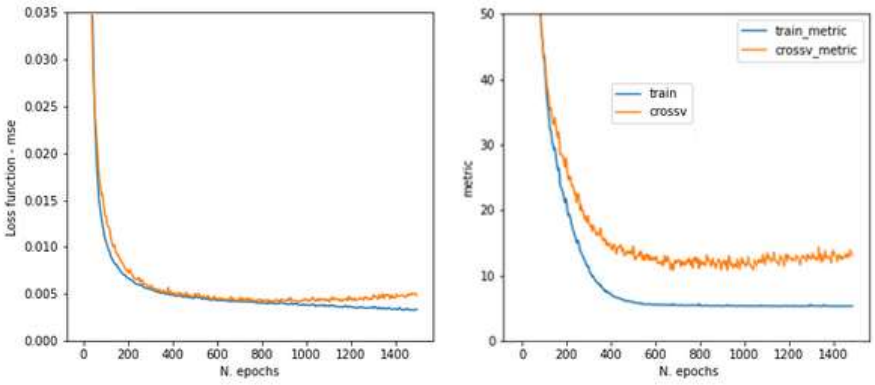


Figure 18. 6L model. Loss function and metric results

As can be seen in Figure 18, after 1000 epochs the training error keeps on descending while the crossv starts to increase. This is the characteristic behavior of an overfitted state: the model is fitting so much to the training set that it is not able to make good predictions when evaluated upon a different set. This leads to an increase in the "generalization error" of the model.

This model was the one used to evaluate the other hyperparameters including the application of a regularization technique to reduce the overfitting.

4.1.2. Hyperparameters Study

- **Minibatch Size Effect**

The first hyperparameter considered was the mini-batch size. Several values were considered, and the loss function results are presented in Figure 19. The different minibatch sizes were a function of the power of 2 because it gives the best performance in terms of memory allocation [37].

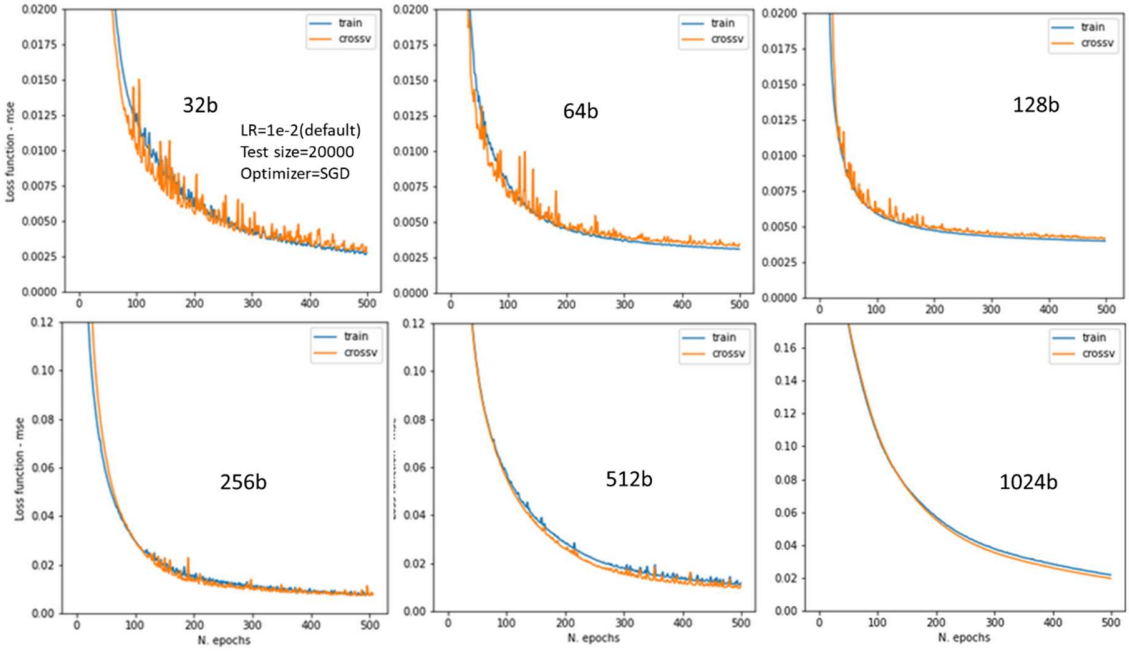


Figure 19. Loss function at different mini-batch sizes

As can be seen in Figure 19, the increasing of the mini-batch size causes two effects. Firstly, the curve becomes smoother, and secondly, the error level increases for the same number of epochs. The "noisy" behavior is characteristic of a network trained with the minibatches method. The lower the size of the mini-batch, the noisier it becomes. This is because parameters are being updated based on less data during each epoch. Each time a minibatch passes forth and back throughout the network, the parameters are updated. The next minibatch will receive those new parameters to calculate the loss function and make another update. This is done until all the minibatches are "seen" by the network. Since the network is not receiving all the training information at once (called Batch Gradient Descent) but in "doses", the loss function can vary considerably between minibatches. The extreme case is when the Minibatch size is 1 and it is called: Stochastic Gradient Descent. This effect is reduced as the minibatch size increases which explains the smoothness of the 1024b curve in Figure 19. On the other hand, the error level is increased also. This imply that more epochs are needed to get an optimum point in the training process.

At this point the first metric drawback was identified. In some of the runs, the metric exhibited an erratic behavior, represented as big noise along its decaying behavior. This was attributed to the fact that it is always looking for the maximum in the whole set, therefore it could switch between samples as training proceed, making the metric curve looking not so well (see the metric noise in

Figure 17 and Figure 18). Another drawback of the VecMax metric was regarding its occasional large values. This too will be discussed later.

- **Learning Rate Effect**

In Figure 20 the results when two different learning rates are evaluated in the model.

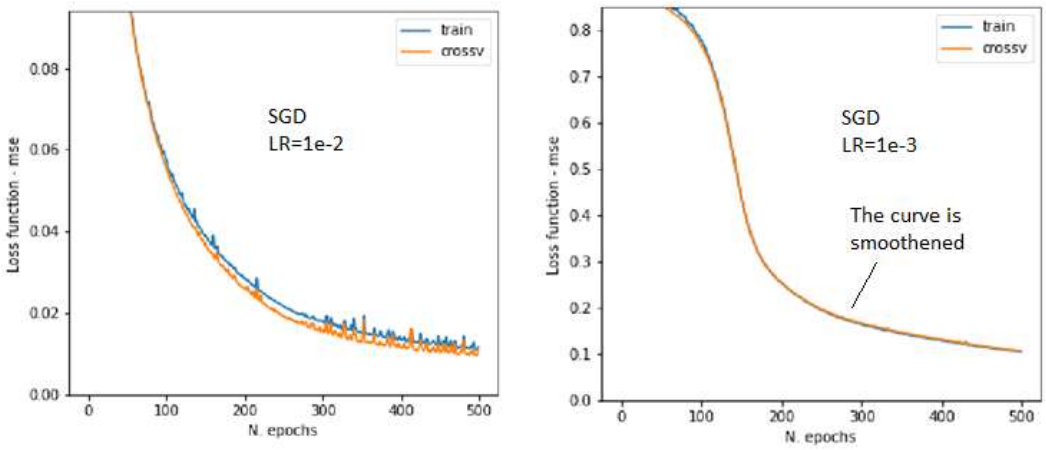


Figure 20. Learning Rate Effect. SGD Optimizer

As it is seen, the reduction of the learning rate brings a direct reducing effect on the noise. This hyperparameter controls how much the weights are adjusted with respect to the gradient during training. This factor can be thought of as the step size that the parameters take between iterations on its way towards the loss function minima. A reduction of this step size means a smaller change in the weights between updates which causes the loss function to look smoother. Consequently, the training would take more iterations to get an optimal solution. (see Figure 21)

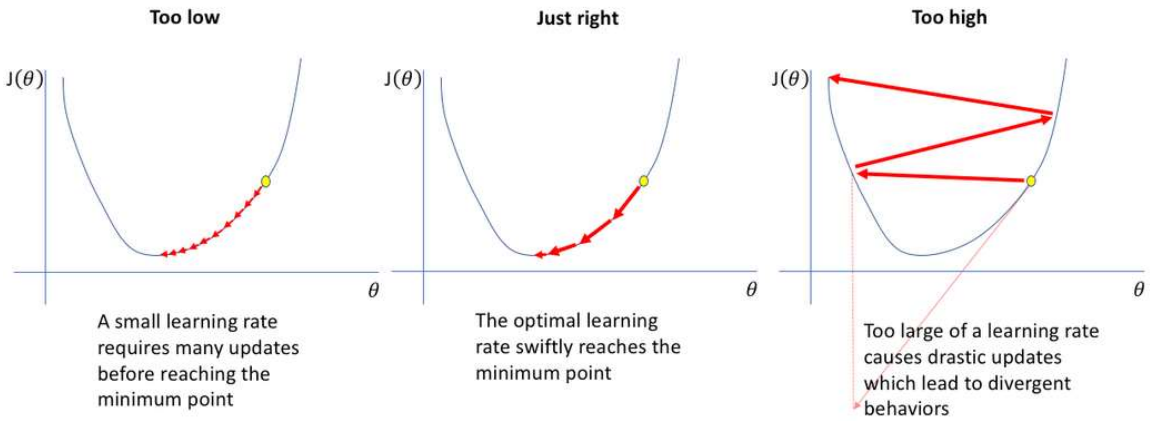


Figure 21. Learning rate schematic representation. (Source: Jeremy Jordan's blogpost [38])

- **Optimizer Effect**

The different optimizers evaluated are presented in Table 9. A detailed description of each of those optimizers can be found in references [39] and [40]. The tests were made over the 6L model with 800 epochs and with a Batch Gradient Descent scheme (Minibatch size = Number of samples). A

train set size of 20000 and a cross validation set of 2000 samples were used. The learning rate of the first two optimizers was increased on purpose because with a LR=1e-4 the loss function had a negligible reduction for the number of epochs chosen (It practically stood still at 0.89).

Table 9. Optimizers Evaluated.

Legend Name	Learning Rate	Optimizer Name
SGD	1E-02	Standard Gradient Descent
SGD Mom	1E-02	SGD with momentum (β=0.9)
RMSProp	1E-04	Root Mean Square Propagation
AD (LR=-4)	1E-04	Adaptative Momentum (Adam)
AD (LR=-5)	1E-05	

In Figure 22 are shown the results when evaluating the model with different optimizers.

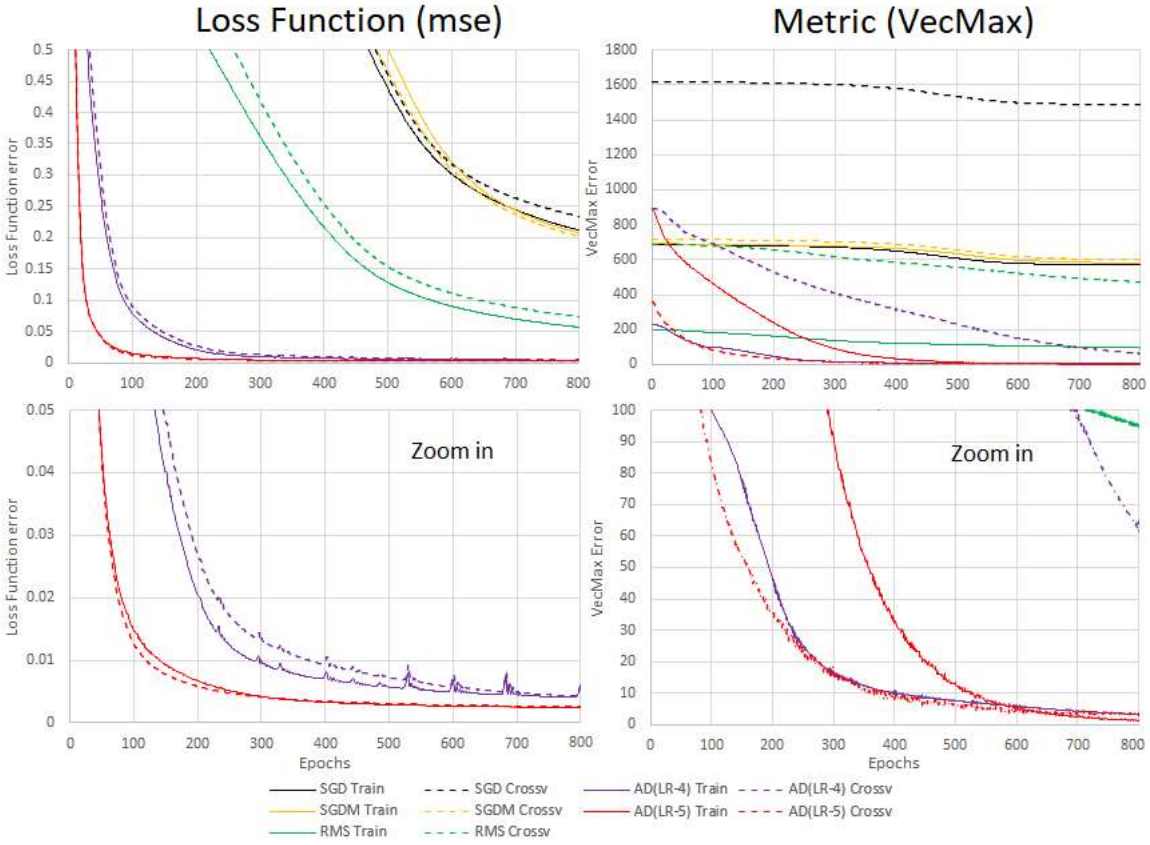


Figure 22. Loss Function (Right) and Metric (Left) results for the evaluated optimizers.

As can be seen in Figure 22, the behavior of the curves, both for the loss function and for the metric, is directly related to the kind of optimizer used. The lowest convergence rate was obtained for the SGD optimizers for which it was even necessary to increase the learning rate to get a noticeable variation in the training curve. On the contrary, the largest rate of convergence for the first 200 epochs was obtained for Adam which belongs to the family of adaptative learning rate optimizers, surpassing even the RMSprop which is part of the same family. The adaptative characteristic gives

to an optimizer the ability to vary the weights and biases learning rate in such a way that the model can reach the loss function minima faster [39], [41]. Due to the presence of noise in the first Adam curve (LR=1e-4), the learning rate was decreased to 1e-5, which not only smoothened the curve, but it reduced even more its error level; suggesting that this probably is (or perhaps close to) the optimal learning rate value.

Although RMSprop has a greater loss function error reduction than the SGD optimizers, the metric does not present a significant variation in none of them for the epochs considered. Therefore, the metric is not effective to give additional insights about the rest of the model predictions, so this issue can be counted as the other metric's drawback. On the other hand, for the Adam case, the metric has a significant reduction, which allows to conclude that the training has a significant effect over all the samples. This will result in better model predictions.

The final error obtained with each optimizer at 800 epochs is summarized in the Table 10 and Figure 23.

Table 10. Final loss function and metric error at 800 epochs for each evaluated optimizer.

Optimizer	Learning Rate	LossF Train	LossF Crossv	Metric Train	Metric Crossv
SGD	1E-02	0.2116	0.2340	570.6	1487.7
SGD Mom	1E-02	0.2073	0.2020	580.0	598.5
RMSProp	1E-04	0.0569	0.0735	95.5	473.5
AD (LR=-4)	1E-04	0.0061	0.0062	3.4	61.0
AD (LR=-5)	1E-05	0.0024	0.0026	1.2	3.1

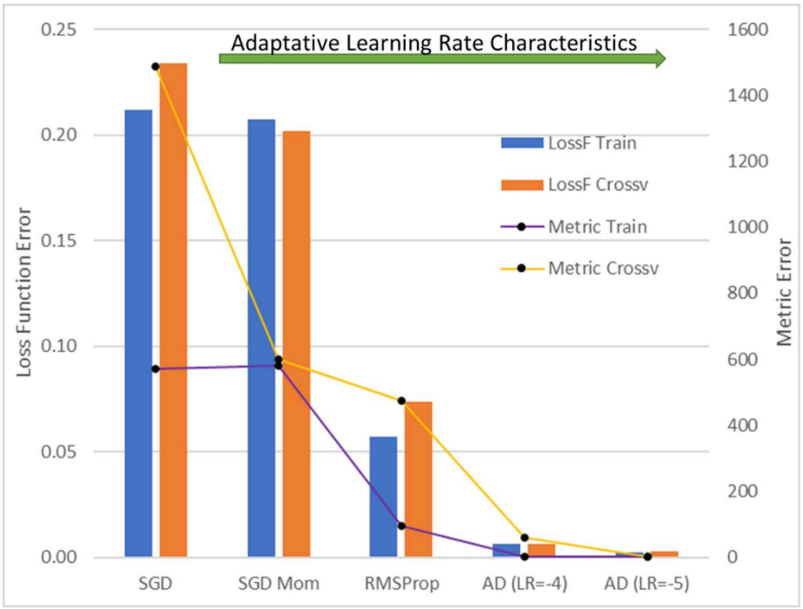


Figure 23. Bar chart of the error for each evaluated optimizer.

The results of Figure 23 shows the error reduction as the optimizer gets more "adaptative" characteristics. From the SGD optimizer that has no adaptative learning rate, until Adam that has fully adaptation in the weights and biases.

For the Adam case, not only the loss function presents a lower error, but also the metric, where there is a substantial reduction in the bias and variance when comparing to SGD (see Figure 22, *Upper Right*). The best scenario was again achieved by the Adam optimizer with a learning rate of 1e-5.

Based on the results of Figure 22 and Figure 23, the optimizers can be sorted according to the error obtained. From the largest to the lowest error, the sequence is:

$$SGD > SGD\ Momentum > RMSProp > Adam$$

- **L2 Regularization**

As the name suggests, the L2 Regularization is a technique used to regularize the parameters of a model by adding a special term to the loss function. This term corresponds mathematically with a factor, called the lambda factor ($\lambda \geq 0$), that multiplies the Frobenius norm of the weights matrix.

$$Frobenius\ Norm: \|w\|^2 = \sum_i \sum_j \left(w_{i,j}\right)^2 \qquad (7)$$

It has the specific effect of penalizing the weight matrices from being too large in the updating step. In practice, the regularization reduces the model's tendency for overfitting. For a deeper explanation about this and other regularization techniques refer to [42] and [43] references.

The evaluation of different lambda values over the 6L model are shown in Figure 24.

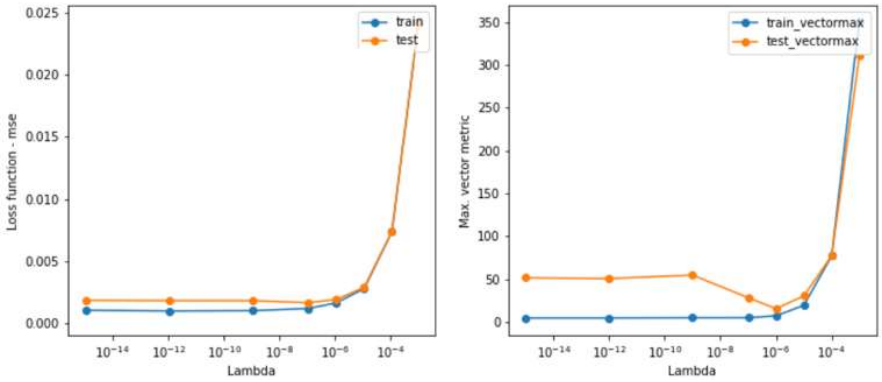


Figure 24. L2 Regularization effect. Loss function (left), Metric (right).

As can be seen in Figure 24, at a lambda above λ = 1e-9, the variance starts to be affected by the regularization term. The metric curves give even better insights about what is happening within the model. At low values of lambda ($\lambda \leq$ 1e-9), the effect is practically negligible, so the overfitting remains the same. For lambda values close to one, on the other hand, the decaying effect upon the weights is so large that many nodes are cancelled out completely, and the model starts to underfit the training set. This issue translates in a bias increase. However, between λ=1e-9 and λ=1e-5 there is a region where the variance diminishes without affecting the bias. This region has a minimum at 1e-6 which corresponds with to the most suitable lambda for the model.

- **Train Set Size Study**

Based on the best conditions found in previous discussions, several train sizes were tested upon the model, and the results were compiled in Figure 25. The cross-validation size is 10% of the train set

size. Following the discussion to reduce noise with the minibatch size, Equation (8) was defined as function of the train set size (This equation gives 1024 for a train size of 20000).

$$MiniBatch\ size = 2^{floor[\log_2(Tsize/20)]+1} \qquad (8)$$

Where $floor()$ function gives the nearest integer down.

Figure 25. Train set size study

From Figure 25, it is possible to observe that the cross validation and training curves have a kind of mirrored behavior. For the interval below 5000, the number of samples are so low that the network absolutely overfits the training set. In a real plant problem where the available amount of samples is limited, a regularization technique can be applied in this case to reduce the variance. However, there will be a high risk that the bias error increases drastically, so the model predictions will be less accurate. To overcome this issue, the best alternative always is to increase the number of training samples. By doing so, the network will have more information about the system, making it more robust against set variations. The opposite case is happening above a train set size of 10000. The variance then starts to slightly increase even when the number of samples are significantly high. This is probably because either there are many big outliers in the cross validation set and special data preprocessing must to be applied, or because the model is also overfitting in some way and needs the implementation of a regularization technique (L2, dropout, reduce the size of the minibatches, etc..).

The most important information that can be extracted from Figure 25 is the fact that both curves converge in the interval between 5000 and 10000 samples. This range corresponds to the most suitable train set size range to work with this model without any special treatment. In conclusion, it could be said for the Minimodel, that the minimum number of samples required by the network corresponds to 5000 samples.

In Table Ap 4 of Appendix D, the reader can find some samples picked randomly from the crossv set, to visually compare the results predicted by the Neural Network model versus the ones extracted from the Aspen Model.

4.1.3. Summary

The summary of the process followed for the implementation of the Neural Network upon the Mini model is shown in Figure 26.

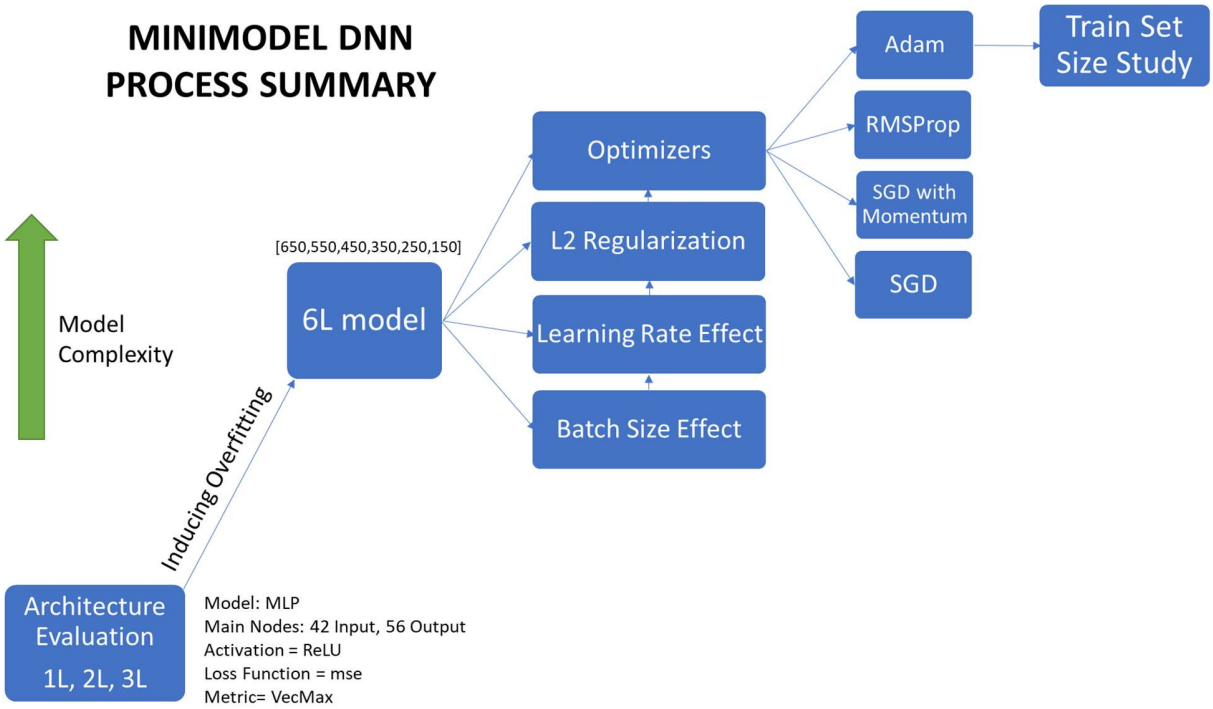


Figure 26. Minimodel Neural Network Implementation Summary.

Finally, the results and discussions given in Sections 4.1.1 and 4.1.2 are compiled in the next general remarks:

- Batch size effect: Increase the Batch ⇒ Reduces noise, increases the error level. More epochs are needed to get a stable point.
- Learning rate Effect. Decrease the learning rate ⇒ Reduces noise. Sometimes are required more epochs to get a stable point. The error level can reduce or increase slightly depending on the optimizer.
- Optimizer: The best optimizer is Adam in terms of error level and rate of convergence. The sequence from the best to the worst ⇒ Adam > RMSprop > SGD Momentum > SGD.
- Train Set Size: The most suitable train set size range to work with the Minimodel network is between 5000 and 10000.

4.2. Full IGCC Model Results

4.2.1. New Metric Definition

Taking all the experience acquired in the Mini model Study, the first network architecture tested was the 6L model over the results obtained from the Full IGCC Aspen model. However, given the drawbacks found in the VecMax metric, another more useful metric was implemented for this case.

Based again on the multidimensional vectorial concept presented in Section 3.3.6, a new metric was derived. Bringing back the definition of the resultant vector $\vec{Y}_{res}$, its magnitude can be thought of as a measure of the error between the expected and the predicted hypervectors ($\vec{Y}_{exp}$ and $\vec{Y}_{pred}$ respectively). The greater the magnitude, the greater the error. Hence, the sum of all the magnitudes of the resultant vectors of the set can be seen as a measure of the total error of the network with respect to that set (Train or Crossv):

$$\begin{aligned} |\vec{Y}_{res}| &\rightarrow measure\ of\ the\ Error \\ \sum_m |\vec{Y}_{res}| &\rightarrow measure\ of\ the\ total\ Error \end{aligned} \quad (9)$$

Where m corresponds to the total amount of samples in that set. If the measure of the error (i.e $|\vec{Y}_{res}|$) is added to the magnitude of the expected vector (i.e $|\vec{Y}_{exp}|$), it will give a quantity that is related on how much the expected vector is increased by the error present in that sample. At the same time, if that quantity is summed over all the samples and divided by the total sum of the expected vectors of the set, an estimation of the total relative change of the expected samples it will be obtained. Expressing all in mathematical form, a new function that was called the Total Variation Increase (TVI) is defined by equation (10).

$$Total\ Variation\ Increase\ (TVI) = \frac{\sum_m (|\vec{Y}_{expected}| + |\vec{Y}_{res}|)}{\sum_m |\vec{Y}_{expected}|} \quad (10)$$

Now, the new metric was defined by inverting equation 9:

$$New\ Metric\ ("VecAcc") = \frac{1}{TVI} = \frac{\sum_m |\vec{Y}_{expected}|}{\sum_m (|\vec{Y}_{expected}| + |\vec{Y}_{res}|)} \quad (11)$$

This new metric is a function with a range $\in (0,1]$, where a value close to 0 will corresponds to a network that makes very poor predictions, whereas a value close to one will mean that the predictions are close to its expected values. Hence this metric can be seen as a kind of "accuracy" of the Network and, from now on, it will be referred to as the "VecAcc" metric.

The VecAcc metric avoided the problem found on some metrics, regarding for instance dealing with the variables close to zero (see Section 3.3.6). Instead of only following the maximum error as the VecMax, the new metric gave a better description of the state of the Network as whole, allowing to make comparisons between different neural network models. The difference between the old and the new metric for the 6L model applied over the Full IGCC data can be seen in the results presented in Figure 27. The overall conditions of the model are presented in Table 11. The total amount of data

extracted from Aspen to implement the Network was 52023 samples divided in a Train:Crossv:Test ratio of 80% : 10% : 10%.

Table 11. Neural Network starting conditions for the Full IGCC model

Loss Function	mse
Activation Function	ReLU
Optimizer	Adam
Learning Rate	1E-05
Data Preprocessing	Stnd
Train set size	43353
Cross Validation set	4335

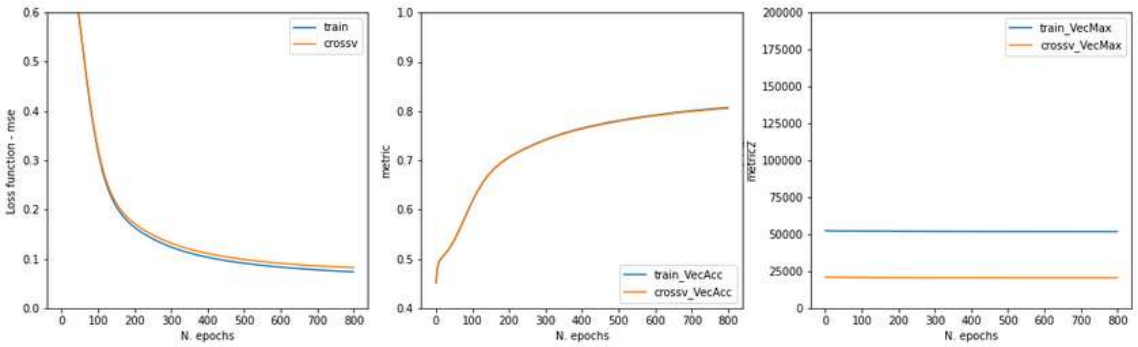


Figure 27. Metric Comparison over the 6Lmodel of the full IGCC case.

As can be seen in Figure 27, the old metric (VecMax) just shows what happens with the maximum error, and in this case found itself stuck on a large value (see section 4.1.2), giving no useful information about the other sample sets. On the contrary, the new metric (VecAcc) presents a consistent global picture of the Network and tells us also that this network has an "accuracy" of: 0.8074 for Train and 0.8058 for Crossv data sets. Those values can be used as comparison factors between different models.

Hence, the new metric and the loss function will be the base quantities upon which the other models will be compared. This will be done along the neural network optimization pathway that will be described in the next sections.

4.2.2. Starting Architecture Evaluation

After the 6L NN model was implemented upon the Full IGCC model data, a slight increase in the number of nodes of the last layers was performed to see if the network error would decrease or just overfit the data. The final values of the models are presented in Table 12.

Table 12. Comparison 6Lmodel with a more node-expanded model for the full IGCC case.

	Loss Function		**Metric (VecAcc)**	
Neural Network Model	Train	Crossv	Train	Crossv
6L-[660,550,450,350,250,150]	0.074	0.083	0.807	0.806
6L-[660,550,450,350,350,350]	0.065	0.076	0.824	0.821

Since the Crossv error still decreased, it was decided to proceed with the same "overfitting induction" method that was applied for the Mini model study (see Section 4.1.1). Therefore, a 10-layer model was implemented with the conditions and results presented in Table 11 and Figure 28 respectively.

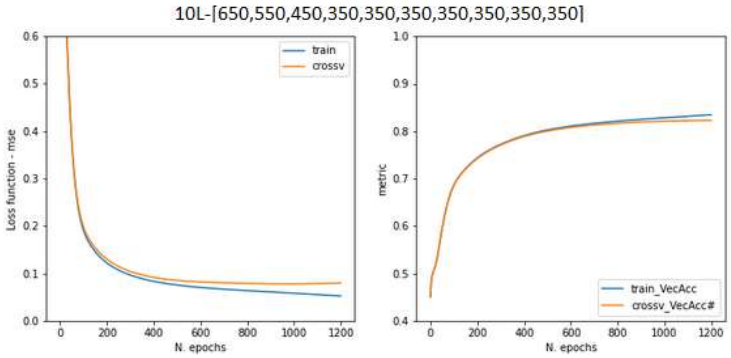


Figure 28. 10-layer Neural Network model for the full IGCC case.

From Figure 28, it can be seen how the new metric responded also to the overfitting state similar to the loss function curve. This confirms once more the usefulness and consistence of the VecAcc metric. Upon this model, from now on referred to as the 10L model, the hyperparameters study will be evaluated.

4.2.3. Loss Function Evaluation

The Loss function is considered as one of the main hyperparameters in the training process. The form of this function depends on how the weights are updated, and it is literally the objective function to be minimized. Hence, in this section several loss functions were tested and compared with respect to the loss given by the *mse* which has been the loss function used so far. Also, for research purposes, a new loss function was developed and tested based on the hypervectors formulation and the metric concept. The function is presented in equation (12).

$$Loss\ Function\ (VecLoss) = \frac{\sum_m |\vec{Y}_{res}|}{\sum_m |\vec{Y}_{expected}|} \qquad (12)$$

This new Loss Function works as a "normalization" of the error with respect to the expected results. So, the best Network fitness is when this function goes to zero. From now on it will be referred to as "VecLoss".

The results obtained for all the loss functions tested is presented in Table 13 and Figure 29. The column labeled "LFerror" corresponds with the respective loss function values at 1200 epochs. The columns labeled "*mse*" are the *mse* values calculated after the training was perform with the respective loss function, so it was not used in the optimization but just for comparison purposes (in the case of the loss function=*mse*, the values in the columns *mse* and LFerror are the same for obvious reasons). More explicitly, the *mse* value was used to make the comparison between the different loss functions tested, the lower the *mse*, the better the loss function. The columns "Metric" corresponds to the VecAcc result.

Table 13. Loss function comparison. Full IGCC case.

Loss Function	Train LFerror	Crossv LFerror	Train mse	Crossv mse	Train Metric	Crossv Metric
VecLoss	0.183	0.192	0.063	0.076	0.845	0.839
mse	0.053	0.080	0.053	0.080	0.834	0.822
Huber	0.015	0.016	0.072	0.084	0.835	0.830
logcosh	0.014	0.015	0.072	0.084	0.835	0.830
mae	0.081	0.082	0.135	0.144	0.764	0.762

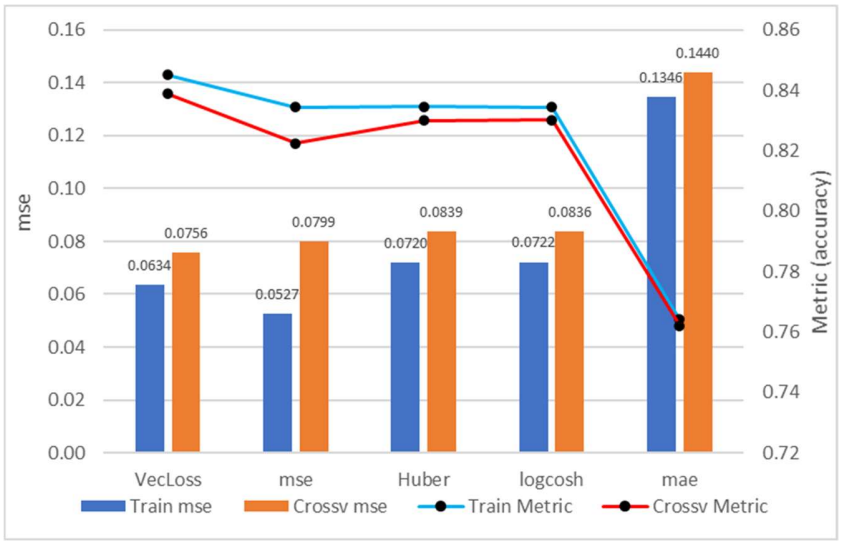


Figure 29. Loss functions comparison. Full IGCC case.

According to the results, in terms of the Crossv mse and the metric accuracy, almost all the loss functions were beaten by the *mse*, except for one. Surprisingly, the new loss function (VecLoss) presented a lower *mse* value for the Crossv (-5.4%) and a greater metric accuracy, than the *mse* loss function itself. From Figure 29, it is possible to observe the regularizing effect that the VecLoss function provided to the model by decreasing the variance (in the metric and *mse* value). This can also be corroborated from Figure 30 (upper left and lower left), where the overfitting was not only reduced, but also the point of appreciable divergence in the metric (Figure 30 - Middle), was shifted a significant amount of epochs (from ∼700 until ∼1000 epochs, check also Figure 30 - Right). This represents a great advantage for the new loss function.

The results show that VecLoss performed better than *mse* in several respects. It is likely that the surface formed by VecLoss, is enhancing the ability of the optimizer to drive the parameters on its way downwards to find a more optimum point than the *mse* function. The "normalization" characteristic that VecLoss function have over the network error (See equation (12)), might be playing an important role in this process.

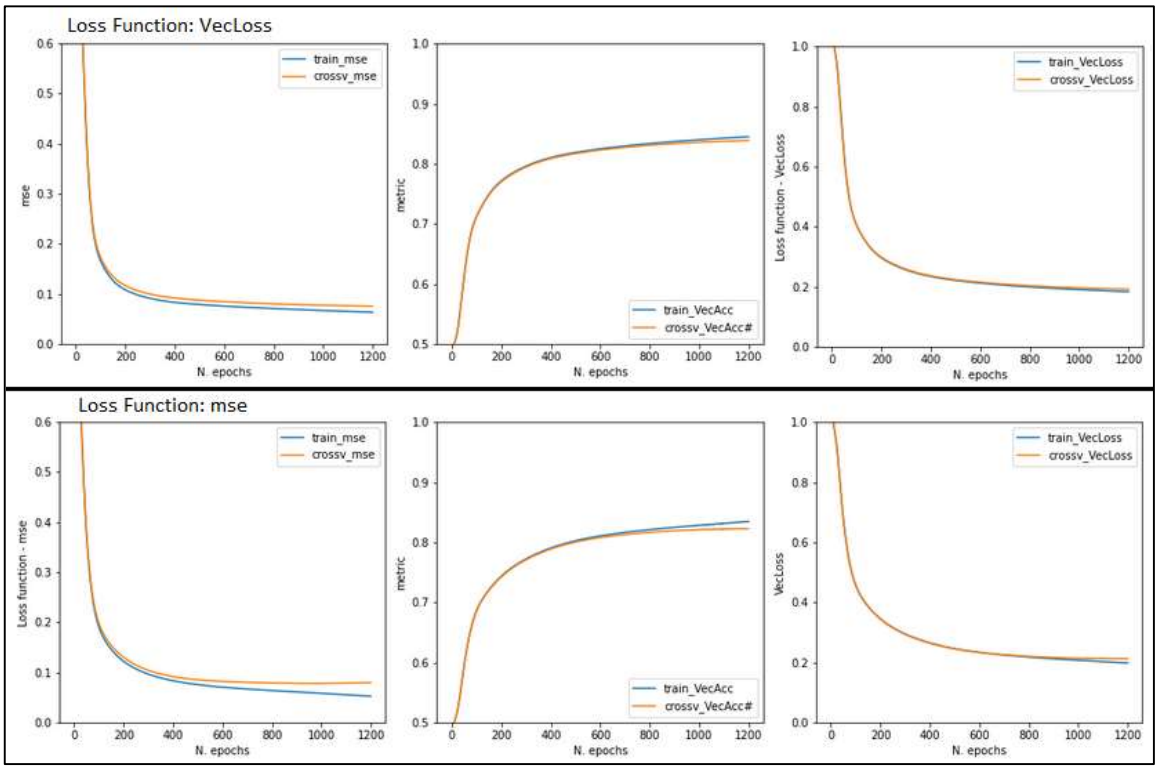


Figure 30. Results comparison between VecLoss (up) and mse (down) as Loss Functions.

In overall, from the worst to the best, the loss functions can be rated in the following way:

$$mae > (Logcosh \approx Huber) > mse > VecLoss$$

Hence, the VecLoss will be the Loss function that will be used for the next models.

4.2.4. Activation Function Evaluation

Another very important parameter that affects the Network training process is the activation function. Again, several activation functions were tested, and its results are presented in Figure 31. A graphical and mathematical description of those functions can be found in Appendix C.

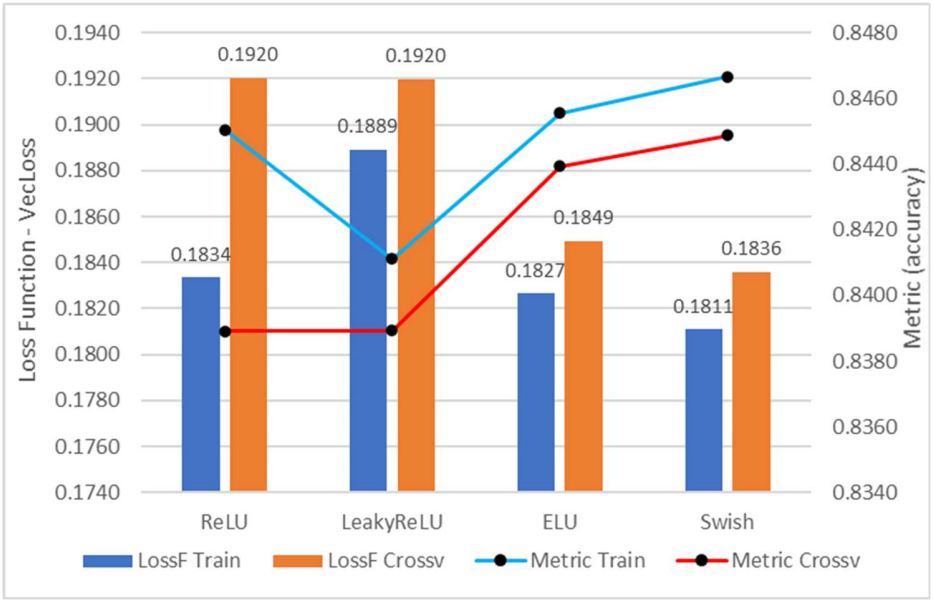


Figure 31. Activation Functions results comparison. Full IGCC case.

From these results, the lowest error was given with the Swish function. Also, the variance (the gap between the Train and the Crossv curves) obtained with the ReLU function was significatively reduced when the other activation functions were implemented. In general terms, the functions that are not strictly zero for the negative values of Z_n (the linear transformation performed in each node depicted in Figure 4), and that additionally are taking a non-linear variation at that range, were the ones that performed the best (ELU and Swish, see Appendix C).

For the activation functions, taking the variance and Crossv accuracy, from the worst to the best error, the sequence is:

$$ReLU > LeakyReLU > ELU > Swish$$

Therefore, Swish was selected as the activation function to be used.

4.2.5. Network Topology Evaluation

So far, the only model evaluated has been a 10L model with the nodes arranged in the order: [650,550,450,350,350,350,350,350,350,350]. This is called a "decremental topology", "10L_dec" in this case. In order to evaluate the influence that the "topology" had over the results, an "incremental topology" was implemented. A new 10L model was defined as: 10L_[200,250,300,350,400,450,500,550,600,650] or in short: "10L_inc". The results were evaluated after 1200 epochs and are shown in Table 14.

Table 14. Loss function and metric results for an incremental and decremental network topology.

	Train		Crossv	
Model	**LossF**	**metric**	**LossF**	**metric**
10L_dec	0.198	0.835	0.208	0.828
10L_inc	0.175	0.851	0.197	0.835

From Table 14, shows that the incremental topology performed slightly better for both the loss function and the metric. This is probably due to the expansion of parameters as it is going to deeper layers, which create more connections between the nodes of the last hidden layer and the output layer. Therefore, each single node of the output layer was defined by many parameters which results in a better description of the non-linearities present in the final result. More incremental and decremental topology architectures will be evaluated in Section 4.2.7.

Hereafter, the Neural Network model used was: 10L_inc.

4.2.6. Data Transformations Effect

At this level, a checkpoint over the Network results was performed. Before entering the data into the Network for the training process, all the samples were previously standardized as a preprocessing step to statistically make the variables centered around zero and with standard deviation equal to 1. This transformation works as a scaling of the variance, and avoids that those variables with large variance dominate the loss function, making the network less able to learn from other variables correctly. Hence, as a matter of verification, the checkpoint consisted of taking the predictions of the last Network model and invert the standardization to get back the "actual" values of the output variables (i.e. the temperature, pressure, etc..).

It was found that while some variables fit very well, others did not over many of the sample sets. In Figure 32, a handful of sample sets are presented for some of the output variables. The units of the variables presented are all in kg/hr.

	Good		Good		Bad		Good		Bad	
	GASFR_SYNGAS-B_H2		CLEANING_COND1_COS		GASFR_SYNGAS-B_CH4		METHANIZ_HICH4_CH4		NH3_VENT_H3N	
Samples	Predicted	Expected	Predicted	Expected	Predicted	Expected	Predicted	Expected	Predicted	Expected
1	9858.16	9856.04	5.04	5.03	4215.52	4325.99	34417.57	34755.91	43.59	68.30
2	11080.89	11064.09	4.11	4.08	4679.06	4484.97	28935.29	28953.53	35.89	30.13
3	9140.70	9119.99	5.96	5.93	1017.75	570.74	11009.03	10810.83	218.16	84.44
4	11775.19	11745.34	3.80	3.85	1120.48	1076.08	23041.69	23129.59	-6.23	4.62
5	8658.10	8564.48	6.27	6.25	-208.70	2.93	11099.42	11402.93	0.94	3.18
6	10193.32	10116.01	4.91	4.85	3926.97	4167.00	902.75	1650.75	460.35	842.66
7	6423.36	6395.14	8.82	8.95	-115.27	0.35	22706.67	22407.30	33.58	47.54
8	10160.21	10129.88	4.77	4.79	4288.82	4647.57	18790.04	18959.21	1.24	1.88
9	6098.63	6178.88	9.14	9.16	33.39	0.35	18507.26	18468.28	6.20	14.53
10	8930.14	8955.26	5.91	5.92	1792.03	1820.94	14112.15	14287.95	6.80	1.41
11	7339.31	7329.36	7.77	7.67	-36.30	5.00	13555.71	13788.73	1.21	8.74

Figure 32. Examples of output variables that have a good and bad fit to the data.

This challenging problem required a deeper understanding about the statistical nature of the data the network was trying to predict. A clever way to solve this "bad-fitting" issue, was by checking the distributions of both the bad and well-fitted variables. The histograms generated, revealed that particularly the bad-fitted variables presented an exponentially decaying distribution (and some of them very skewed), while the good-fitted ones described a more gaussian-like distribution. The distributions were not affected when the standardization was applied, only the magnitudes were. Some examples can be seen in Figure 33.

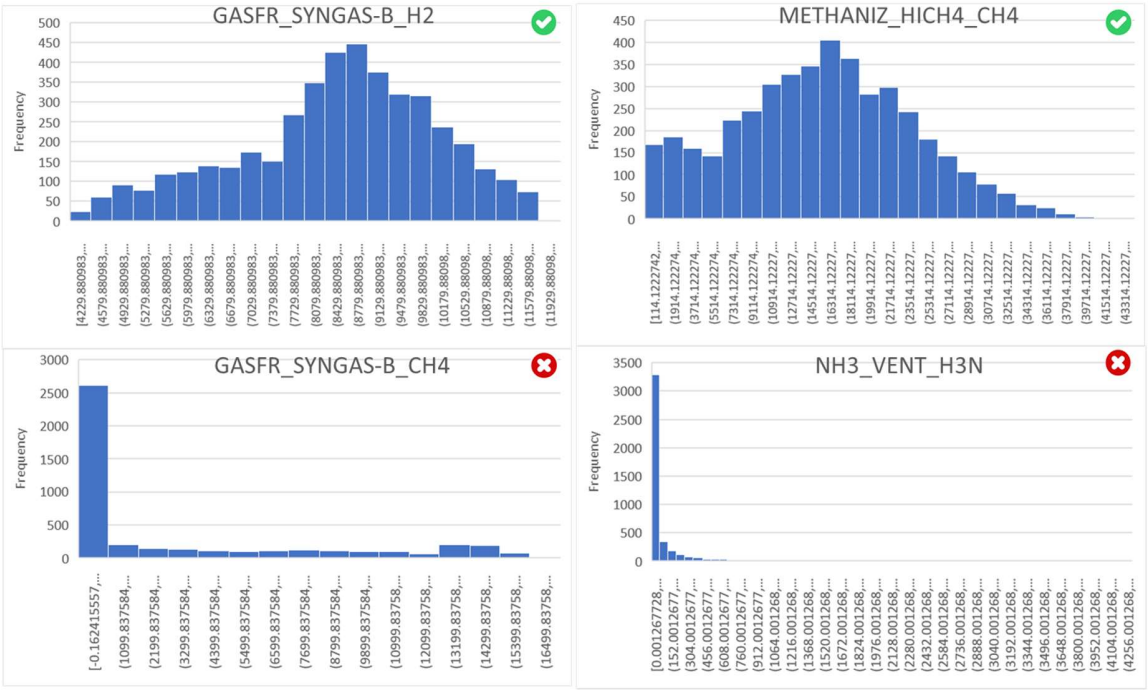


Figure 33. Statistical distributions of the good and bad fitted Variables.

To make the analysis short, the variables that were poorly described by the network contained many outliers in the data. These outliers caused a huge distortion and skewness in the distribution, so the network was not able to fit the whole samples' landscape in a good way. Just the large values of the bad-fitted variables were predicted more or less well (see for instance, rows 8 and 9 of the GASFR_SYNGAS-B_CH4 variable in Figure 32).

The straightforward approach to solve this problem, was to try to transform the data in such a way that the distribution of the bad fitted variables was more like the good-fitted ones, i.e. more gaussian-like (see Figure 33). Among the candidates considered were: Logarithmic, Power, and Quantile Normal transformation. The logarithmic and power transformation consisted in applying a logarithmic and power function (with an exponent less than one) to the data, respectively. Their results are not shown here because they were not satisfactory, although they improved for some of the bad-fitted variables.

The Quantile Normal Transformation (here referred to as Qnorm) is based on the Quantile function formulation, sometimes called Percent-Point function (PPF), which corresponds to the inverse of the Cumulative Distribution Function (CDF) of a variable. By using the quantiles information of a variable's data, the quantile function transforms any distribution the variable has into another distribution. For the Qnorm it will be a Gaussian distribution. In practical terms, this kind of transformation aims to spread out the most frequent values in a variable, thus reducing the impact of the outliers. Additionally, the Quantile Uniform transformation was also applied (here referred to as Qunif). Unlike the Qnorm, this transformation sets all the quantiles at the same frequency making the distribution more uniform. Figure 34 shows the effect of the Qnorm and Qunif when applied over the data. The transformations can be found as the "QuantileTransformer" class within the scikit-learn library for Python (see reference [44] for more information).

These transformations are not so widely used (nor taught) as the standardization, but they are also part of the possible preprocessing steps that can be performed upon the data (see reference [45] p.15).

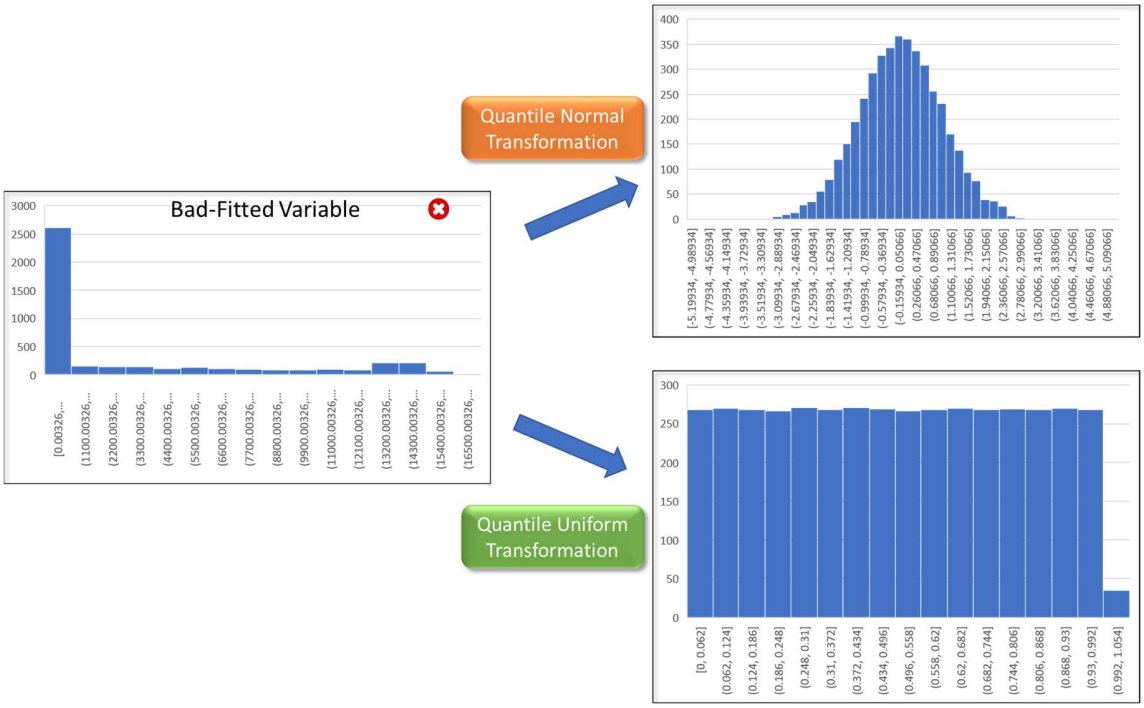


Figure 34. Effect of the Quantile Transformations over distribution of a variable

To compare the results of the transformations two new quantities were calculated: The Average Error Ratio per Sample (AERS) and the Average Error Ratio per Sample per Variable (AERSV). Those quantities were defined according equation (13) and (14).

$$AERS_j = \frac{1}{m}\sum_{i=1}^{m}\left|\frac{Y_{i,j}^{acpred} - Y_{i,j}^{acexp}}{Y_{i,j}^{acexp}}\right| \qquad (13)$$

$$AERSV = \frac{1}{V}\sum_{j=1}^{V}\left(\frac{1}{m}\sum_{i=1}^{m}\left|\frac{Y_{i,j}^{acpred} - Y_{i,j}^{acexp}}{Y_{i,j}^{acexp}}\right|\right) \qquad (14)$$

Where V is the total number of output variables (V=335) and m the total number of samples of the evaluated set (Train= 43353, Crossv= 4335, respectively). $Y_{i,j}^{acpred}$ and $Y_{i,j}^{acexp}$ correspond to the predicted and expected "actual" values respectively (i.e. after inverting the transformation), of the j-th variable at the i-th sample.

At the end, four transformations were compared with the AERSV quantity. Equation (14) was manually calculated in an excel sheet with the results of the train set. The AERSV, loss function and metric results can be seen in Table 15. Stnd corresponds to the standardization and MaxMinScaler is the normalization of the values between 0 and 1.

Table 15. AERSV results for the different transformations tested.

	AERSV	
Transformation	**Train**	**Crossv**
MinMaxScaler	50.398	51.000
Stnd	33.579	34.095
QNormal	0.424	0.602
QUniform	0.323	0.323

As is seen in Table 15, the difference in the AERSV values is absolutely conclusive. The Quantile transformations outperformed greatly over the standardization and the MaxMinscaler transformation. The reduction in AERSV error was of 2 orders of magnitude.

The AERS value was also calculated and the results compiled in Figure 35. Only the results of the Qunif and Qnorm are shown because the ones obtained with standardization were exceedingly large and hindered the other ones. The number and the variable correspondence can be found in Table Ap 7, Appendix E.

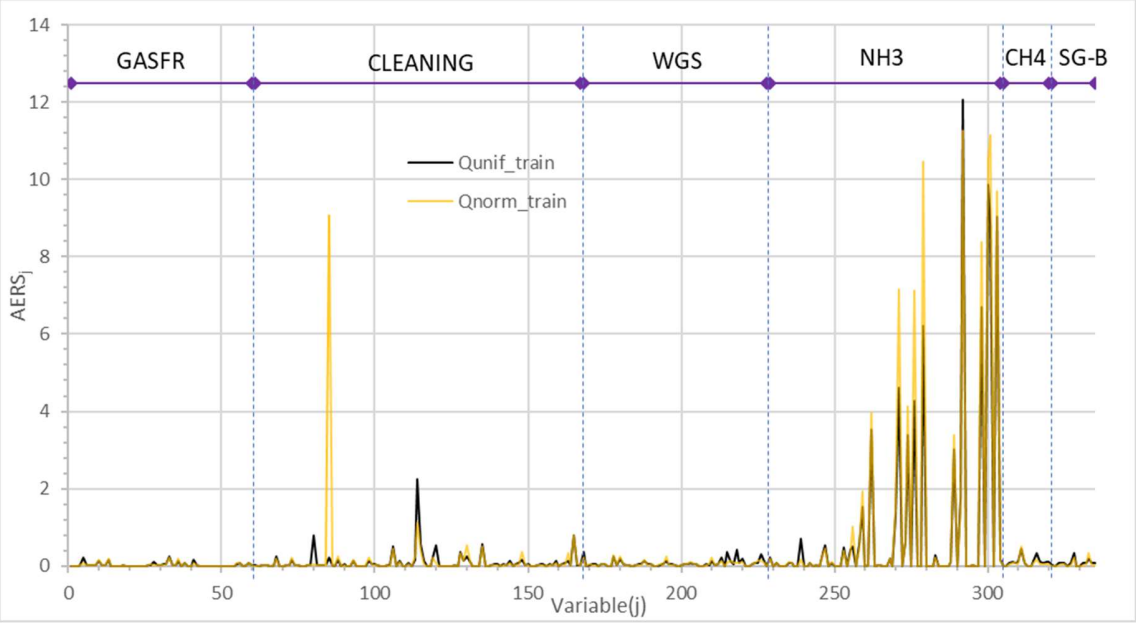


Figure 35. ARSV values for the output variables collected in the full IGCC model.

A closer look at the variables, revealed that the ones that had the largest Average Error Ratio per Sample (AERS) were mainly located at the NH3 block (see Figure 35). In this case, the Quantile Uniform transformation case results corresponded with the AERS bottom limit for most of the variables, including the "bad" ones. No other transformation was found that could go below that limit. This error barrier was attributed to convergence and/or tolerance issues in some units of the NH3 block of the Aspen Plus model (and in the other blocks that presented a significatively large AERS value). The presence of columns, flash units and even heat exchanger units coupled in recycled streams (as the NH3 block case, see flowsheet in Appendix A), etc. can behave as sources of convergence issues due to tolerance parameters assigned in the Aspen simulator. This introduced a sort of randomized effect on the simulation outcome that the network was not able to capture

during its training. This hypothesis was corroborated after several samples were introduced manually in Aspen; consistently, almost all the variables with the largest AERS also presented a completely different value after running the simulation this second time.

Taking in consideration the AERS results for all the variables excluding those of the NH3 block (based on the convergence issues discussed previously), the percentage of the variables with a prediction accuracy above 95% was determined. The results for both the Mini model and the Full IGCC model are given in Table 16. ($Average\ Accuracy = (1 - AERS) * 100$)

Table 16. Percentage of variables and its prediction accuracy for the Minimodel and Full IGCC model.

	Average Accuracy	
Model	**> 95%**	**< 95%**
Mini Model	96%	4%
Full IGCC Model	90%	10%

As seen in Table 16, the percentage of variables of the Full IGCC model with an accuracy above 95% is significantly high when compared with the Mini model. This is a remarkable result given the complexity of the Full IGCC model. On the other hand, the variables with an accuracy below 95% are probably due to the same convergence problem found with the Aspen simulator.

As a secondary source of error, some variables presented a comparatively large AERS because certain samples had a predicted and expected values close to zero but with a high relative difference between them. This issue was called the "*near-zero variable problem*" and had sometimes a strong impact in the calculated AERS although the responsible values were not significantly high *per se*. Therefore, by finding a way to solve this *near-zero problem*, the 95% threshold of Table 16 could increase even more for the same number of variables. Nonetheless, this issue was left to be solved for a future research work.

The graph of the loss function and metric for the Network with Qunif is shown in Figure 36.

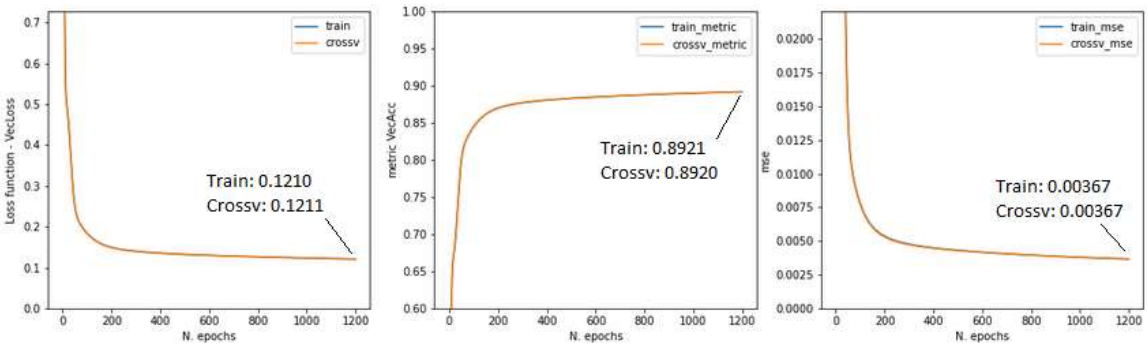


Figure 36. Loss function (VecLoss), VecAcc metric and mse results when Qunif is applied over the data.

As the graph shows, the variance is practically nonexistent in this final model.

Based on the previous discussion and the AERSV results obtained, from the larges to the lowest error, the sequence is:

$$MaxMinScaler > Stnd > Qnorm > Qunif$$

Therefore, the Qunif transformation will from now on be applied as a pretreating step.

4.2.7. Architecture Optimization

To this point, the architecture of the neural network has been fairly deep. However, the results of Figure 36 shows that the Variance, and even the Bias, have been significatively reduced compared to the starting 10L model at the beginning of the process (see Section 4.2.2). Then, it was decided to check whether a smaller architecture can give the same results. A large number of architectures were tested, to identify which was the most suitable one to describe the Full IGCC model. In Table 17 the different architectures tested are presented (at 1200 epochs) along with the total number of trainable parameters they represent. The results of the loss function and the metric against the number of parameters (or Model complexity) are shown in Figure 37 and Figure 38 respectively.

Table 17. Different Architectures tested in the Network optimization.

Name	Model	Nparam
1L_100	1L[100]	47120
1L_350	1L[350]	164120
1L_700	1L[700]	327920
2LInc100	2L[600,700]	733820
3LInc100	3L[500,600,700]	1019620
4LInc100	4L[400,500,600,700]	1205320
5LInc100	5L[300,400,500,600,700]	1310920
6LDec100	6L[700,600,500,400,300,200]	1269920
6LDecflat	6L[650,550,450,350,350,350]	1218070
6LInc50	6L[200,250,300,350,400,450]	725670
6LInc100	6L[200,300,400,500,600,700]	1356420
6LIncflat	6L[350,350,350,450,550,650]	1269970
6LMM	6L[650,550,450,350,250,150]	1033770
6LUnif	6L[400,400,400,400,400,400]	989520
10LInc	10L[200,250,300,350,400,450,500,550,600,650]	2011970
15LInc	15L[200,250,300,350,400,450,500,550,600,650,650,650,650,650,650]	4127720
20LInc	20L[200,250,300,350,400,450,500,550,600,650,650,650,650,650,650,650,650,650,650]	6243470

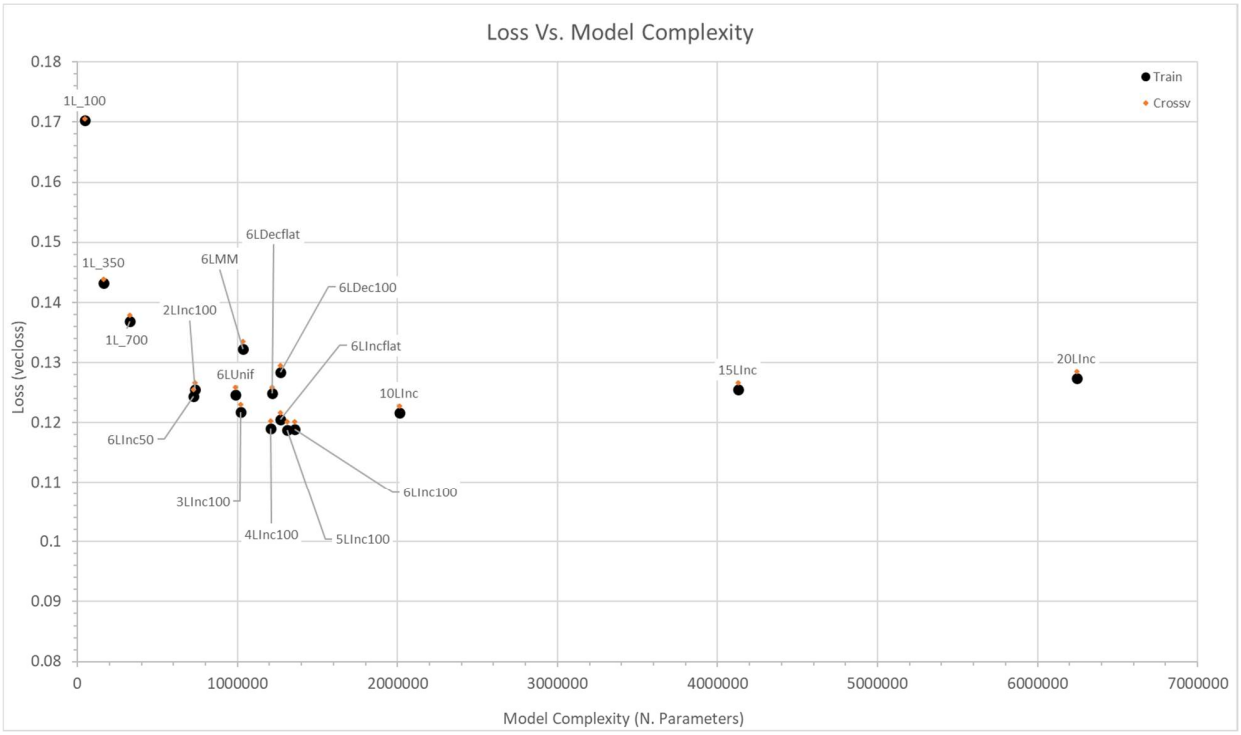


Figure 37. Loss Function Error Vs. Model Complexity. Full IGCC model.

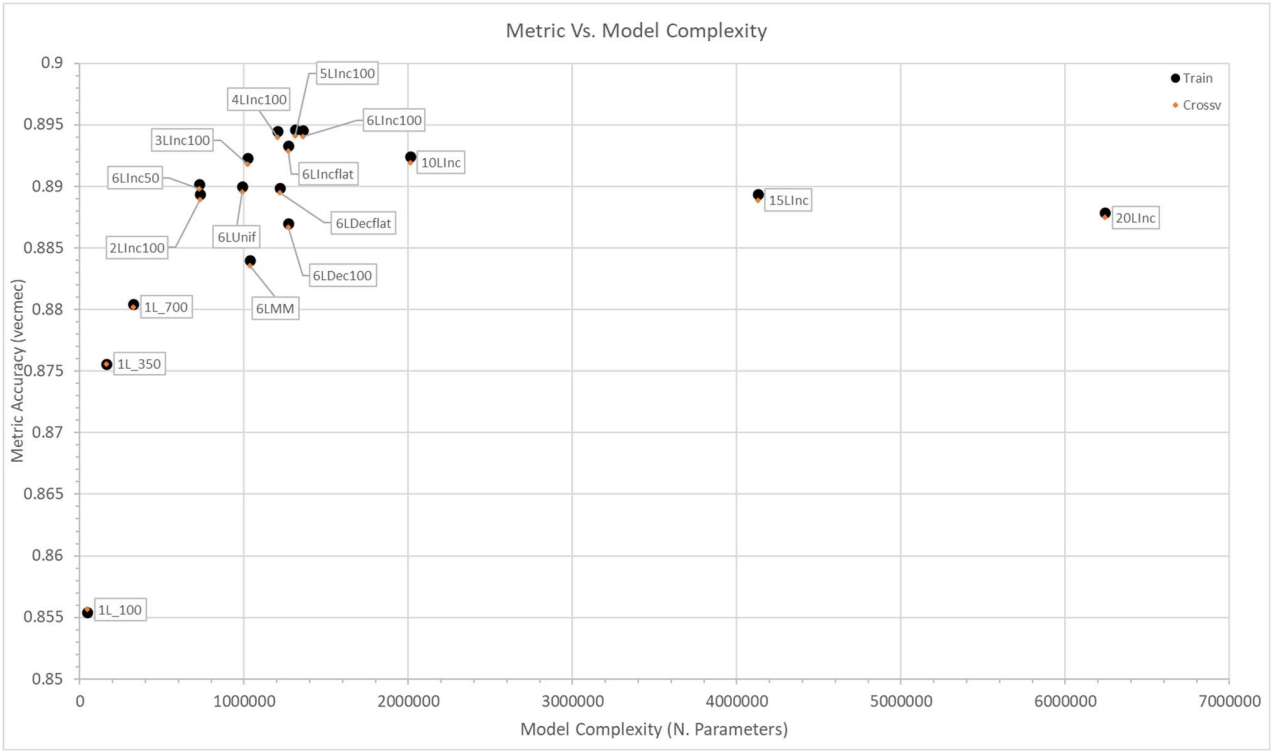


Figure 38. Metric VecAcc Vs. Model Complexity. Full IGCC model.

The Figure 37 and Figure 38 are absolutely compelling. The most relevant observations were extracted and compiled in the following list:

1) According to both the loss and the metric the best architecture was the "**5Linc100"**.
2) For the 6 Layer group, the incremental topologies outperformed once again the decremental ones. Even when the number of parameters were as low as the "6LInc50" model.
3) The uniform architecture (6Lunif) with fewer parameters had better results than several of the 6L group, especially the decremental ones.
4) The "6LMM", which is also part of the decremental group, was the same architecture used for the Minimodel study. This was the worst performing model within the group of deep architectures (> 1Layers).
5) The difference between "2Linc100" vs. "6Linc50" and "3Linc100" vs. "6Lunif" models, shows that the number of nodes and how are they disposed on the layers have an important role in the model performance. Therefore, not necessarily a deeper architecture is given to be more effective than a shallower one.
6) For models with more than 1.2 million parameters, the error started to slightly increase for both the loss and the metric. This is probably due to an issue related to vanishing gradients. Vanishing gradients is an issue that very deep architectures often have when a gradient signal goes to zero quickly, thus making the gradient descent very slow during training. One of the means to solve this problem, is to implement the concept of Residual Networks (ResNet's) [46], [47] but this is out of the scope of this study.

Hence, the final architecture chosen to describe the Full IGCC model was:

$$5Linc100 - [300,400,500,600,700]$$

A validation test was performed over this last architecture and the results are shown in Table 18. The test set size was 10% of the Train set size (the same size as the Crossv set i.e. 4335 samples).

Table 18. Test validation over the final Architecture

SET	Loss F. (VecLoss)	VecAcc	mse
Train	0.1175	0.8947	0.00346
Crossv	0.1182	0.8943	0.00353
Test	0.1184	0..8941	0.00356

4.2.8. Train Set Size Study (Full IGCC model)

Finally, the 5Linc100 model was tested upon several train set sizes and the results at 2000 epochs are presented in Figure 39.

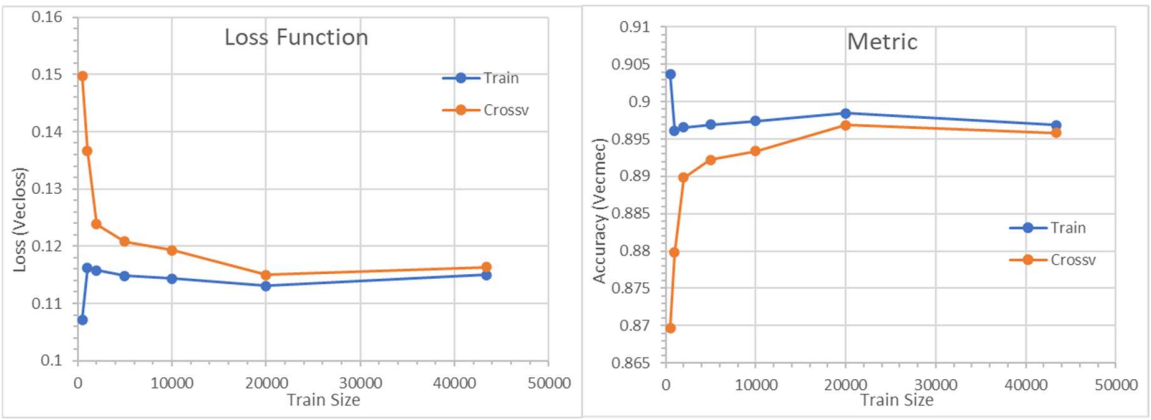


Figure 39. Train set Size Study. Full IGCC case.

As the figure shows, the variance and the bias of the model converge at a train set size of about 20000, which allows to conclude that this is the minimum number of samples that can be used to implement a Neural Network upon the Full IGCC model. This number corresponds to 4 times the minimum train set size required for the Minimodel (see Section 4.1.3), which is an expected result given the magnitude of the system studied.

4.2.9. Summary

The summary of the Neural Network implementation process for the Full IGCC case is presented in Figure 40.

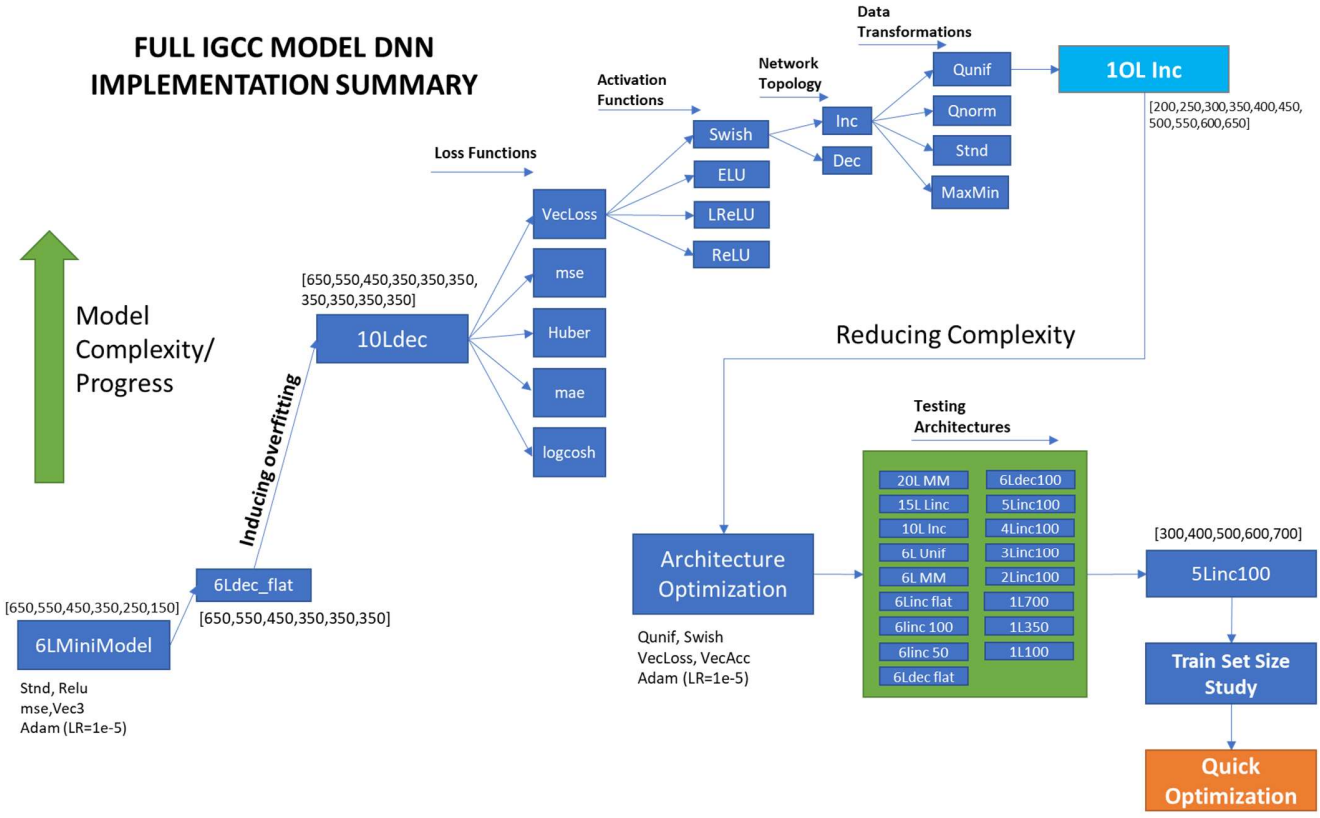


Figure 40. Full IGCC Neural Network implementation summary.

4.3. Quick Optimization Study

One of the expected benefits of a Neural Net model, compared to that of a first principles model, is the robustness and calculation efficiency. These abilities make the model, potentially, suitable for stochastic optimization algorithms that require that thousands of simulations be run in a pseudo random order. Note the difference from the pseudo random generation used in this thesis for the data collection stage (see Section 3.3.2 and 3.3.3), where the data points were first generated and then sorted before being sent to the model, in order to avoid large variations in the input conditions that could lead to convergence errors in the Aspen simulator. This sorting step is not possible in stochastic optimization which is why the success is reliant on a model that can handle large variations in the input space between runs.

To compare the performance in calculation time of the Neural Network model against the Aspen simulator, a quick optimization was done. The total CO2 emissions of the plant was chosen to be minimized. The idea was then to count how many calculations were required to get the optimum value (minimum amount of emissions).

Two optimization methods were applied in order to verify the final result: Particle Swarm Optimization (PSO) and Simulated Annealing (SA). Each of those have their own characteristics and ways of finding the global minima of a function, nonetheless both represent fairly robust methods to perform this kind of tasks. A more detailed explanation can be found in references [48] and [49].

The PSO method was implemented by means of a special Python library called: "pyswarms", whereas the SA was implemented with a code from reference [50]. The results extracted were: 1) The minimum amount of CO2 emissions found by the method for a given fuel rate, 2) The number of calculations, i.e. the number of times the model is being called to perform a simulation, and 3) the time spent in the whole process. The parameters used in both methods and the results obtained are shown in Table 19. The graph of the cost curve for the PSO method is shown in Figure 41 (this is not the same loss as for the Neural Network concept, but it represents the optimum value found by the method after each iteration).

Table 19. Results from the PSO and SA methods in the Quick optimization.

Model	Total CO2 (Kg/hr)	# calc.	Total Time (s)	Calc. Speed (sec/calc)
PSO (particles=50,c1=0.1,c2=0.1,w=0.5)	5453	300	1	0.0033
SA(pstart = 0.7,pend=0.001,m = 50)	5453	6300	235	0.0373

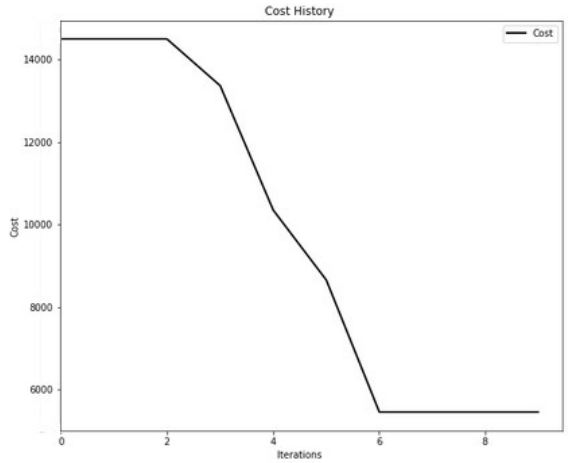


Figure 41. Cost function of the PSO method.

As can be seen in Table 19, while both algorithms reached the same minimum, they required a different number of calculations, and therefore spent a different amount of time to perform the optimization task. Even though the number of calculations can vary between runs and also depends on the parameters used in both models; in a great variety of optimization applications the PSO algorithm outperforms the SA. It is preferred because very often it is more stable and also can reach the global optima in fewer iterations even for incredible non-linear functions [51], [52], [53].

On average, the Aspen plus console took 13 seconds to perform one successful simulation calculation plus another 6 seconds to collect the results. Those values were determined when the data was collected from the Aspen console with the Python interface. Taking the largest time case, the SA algorithm required 6300 simulations before reaching the optimum solution, a task that took 235 seconds to complete with the 5Linc100 Neural Network model. If the same algorithm would

have been applied but using instead the Aspen simulator, the task would have taken more than 33 hours, assuming all simulations successfully completed i.e. without convergence errors. The PSO algorithm required merely 1 second of calculation time using the 5Linc100 model, and would have required 1.6 hours using Aspen Plus, again assuming all simulations successful.

The difference is evident. While the Neural Network model took 235 seconds to perform 6300 calculations (i.e. ~0.037sec/calculation) the Aspen model it would take one day to do the same task.

Although this was a simple optimization task, it is possible to observe the speed superiority of the Deep Neural Networks against a traditional modelling tool; but more importantly, the robustness in large input variations is what really makes a stochastic optimization approach possible with a Neural Networks model. However, in a further research, a deeper and wider optimization can be performed, that could include one or several objective functions with its respective set of constrain equations. Expressed in natural language such a set of objective functions could potentially be of the type: "Minimize the emission levels and energy consumption of the plant while producing the most profitable product mix, all with on-spec purities, while keeping the ammonia production at a minimum of 250 kmol/h". This will allow to test even further the robustness and speed of the Neural Network model over a more realistic scenario as the ones required for decisions-making.

4.4. Future Research Perspectives and Final Remarks

Though it took a lot of time and analysis to get the optimal architecture in this study, once the implementation route has been defined, the making process of a Neural Network model could be converted into a very straight forward routine. Then, once the Network is created, it will be able to perform optimization processes in a really fast and robust way. The plan for a future research work could be focused on:

a) Solve or minimize the convergence and tolerance issues of the Aspen Simulator to reduce the randomness on the poor-fitted variables and improve even more the Network accuracy. (Section 4.2.6)

b) The further refinement of the Neural network implementation process in order to create an even faster route for hyperparameters optimization. (Section 4.2.9)

c) Apply the Neural Network on real plant data. There are processes in operation that have proven notoriously difficult to simulate, such as sulfuric acid production. A neural net type of model could potentially be trained on existing data and then allow for similar optimization schemes as has been presented in this thesis

Finally, based on the results achieved by the Neural network, the potential applications for this type of model can be vast. From the very task of modelling a simulated process until a real facility operation modelling, the Deep Neural Networks have proved to be a potentially very powerful tool to model those kinds of complex and non-linear systems. In future, with more research and refined implementation, it could even be used to give fast and fairly good estimations against any sudden changes of the plant operating conditions, to help plant operators or production managers in their cost-optimal decision-making process.

5. CONCLUSIONS

The present study aims to evaluate the performance of a Deep Neural Network relative to an IGCC Aspen model in terms of time efficiency and accuracy. By setting up a Mini model, the impact of several hyperparameters was identified and the data generation scheme to perform the simulations in Aspen Plus was developed. Finally, a specific hyperparameters tuning process was followed to implement the Full IGCC Neural Network, and its performance in time and accuracy was evaluated. The main findings of this study are summarized in the following list.

- Special random number generators in joint with specific sorting algorithms were tested, to define and adjust the input space that fed the Aspen Model in order to perform the simulations. In terms of lower number of errors and improved neural network performance, the best technique for the data generation was Quasi Monte Carlo with Sobol Numbers (QMCS) combined with Traveling Salesman method modified for hypercubes systems (TSPH) as sorting method. (**Appendix D**)
- The Neural Network for the Full IGCC model was trained using 43353 samples and cross-validated with 4335 samples. A coordinate descent method was applied to perform the hyperparameter optimization process including an evaluation over: 1) starting architecture conditions, 2) Loss functions, 3) Activations functions, 4) Network Topology and 5) Data transformations. Finally, an architecture study was performed and the optimal neural network architecture found for the studied IGCC model was: 5L-[300,400,500,600,700]. (**Section 4.2.7**)
- A Novel Loss function and Metric were developed to evaluate the Network performance. The new loss function allowed to not only decrease the Crossv error by more than 5% respect the *mse* function but also increased the Network stability against overfitting problems. (**Sections 4.2.2 and 4.2.3**)
- After a study of the size of the train set was performed, the minimum number of samples required to build the Neural Network for the Full IGCC model was 20000. This was 4 times more than the one found for the Mini model. (**Section 4.2.8**)
- Generally, about 90% of all collected variables were predicted by the Neural Network model with an average accuracy of more than 95% with respect of the Aspen model results. The variables with an accuracy of less than 95% was shown to came mainly as a result of inconsistent outcomes from the Aspen Plus simulator. (**Section 4.2.6**)
- A quick optimization study was performed by the means of two robust methods (Particle Swarm Optimization and Simulated Annealing), to compare the calculation time between the full IGCC Neural Network model and the Aspen Simulator. The results showed that the Neural Network had a calculation speed of 0.037 sec/simulation, while the Aspen Model took about 13 sec/simulation (excluding data extraction). The simple task of minimizing the CO2 emissions took for the Neural Network between 1 to 235 seconds (depending on the algorithm), while the Aspen model, if at all successful, would require between 1.6 to 33 hours to be completed. (**Section 4.3**)

6. REFERENCES

[1] T. Wang and G. J. Stiegel, Integrated Gasification Combined Cycle (IGCC) Technologies, ELSEVIER, 2017.

[2] J. C. Acevedo and et.al, "Simulation of the gasification process of palm kernel shell using Aspen," *Journal of Physics: Conference Series,* no. 012010, p. 1126, 2018.

[3] [Online]. Available: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/.

[4] [Online]. Available: https://towardsdatascience.com/types-of-neural-network-and-what-each-one-does-explained-d9b4c0ed63a1.

[5] G. Montavon, G. B. Orr and K.-R. Müller, Eds., "Neural Networks: Tricks of the Trade," 2 ed., 2012, pp. 437-478.

[6] S. Kostadinov, "Recurrent Neural Networks with Python Quick Start Guide," 2018, pp. 94-96.

[7] [Online]. Available: https://missinglink.ai/guides/neural-network-concepts/hyperparameters-optimization-methods-and-real-world-model-management/.

[8] C.-H. Cheng, G. Nührenberg, C.-H. Huang, H. Ruess and H. Yasuoka, "Towards Dependability Metrics for Neural Networks," 2018. [Online]. Available: https://arxiv.org/pdf/1806.02338.pdf.

[9] J. Brownlee, "Master Machine Learning Algorithms," pp. 19-21.

[10] [Online]. Available: https://pathmind.com/wiki/comparison-frameworks-dl4j-tensorflow-pytorch.

[11] [Online]. Available: https://towardsdatascience.com/top-10-best-deep-learning-frameworks-in-2019-5ccb90ea6de.

[12] R. Iman, J. Helton and J. Campbell, "An approach to sensitivity analysis of computer models, Part 1. Introduction, input variable selection and preliminary variable assessment," *Journal of Quality Technology,* vol. 13, no. 3, pp. 174-183, 1981.

[13] G. Fishman, "Monte Carlo: Concepts, Algorithms, and Applications," Springer, 1996, pp. 40-42,66-75.

[14] [Online]. Available: https://machinelearningmastery.com/monte-carlo-sampling-for-probability/.

[15] R. Y. Rubinstein and D. P. Kroese, "Simulation and the Monte Carlo Method," Wiley, 2017, pp. 5-6.

[16] S. Glen. [Online]. Available: https://www.statisticshowto.com/latin-hypercube-sampling/.

[17] J. Menčík, "Latin Hypercube Sampling," in *Concise Reliability for Engineers*, 2016.

[18] [Online]. Available: https://en.wikipedia.org/wiki/Latin_hypercube_sampling.

[19] C. Lemieux, "Monte Carlo and Quasi-Monte Carlo Sampling," Springer, 2009, pp. 139-161.

[20] [Online]. Available: https://www.mathworks.com/help/stats/generating-quasi-random-numbers.html.

[21] R. E. Caflisch, "Monte Carlo and quasi-Monte Carlo methods," *Acta Numerica,* pp. 1-49, 1998.

[22] D. L. Applegate, R. E. Bixby, V. Chvátal and W. J. Cook, "The Traveling Salesman Problem: A Computational Study," Princeton, 2006, pp. 2-5.

[23] [Online]. Available: https://en.wikipedia.org/wiki/2-opt.

[24] [Online]. Available: https://kitchingroup.cheme.cmu.edu/blog/tag/aspen/.

[25] [Online]. Available: https://www.tensorflow.org/guide.

[26] [Online]. Available: https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html.

[27] [Online]. Available: https://colab.research.google.com/notebooks/intro.ipynb#.

[28] [Online]. Available: http://colinraffel.com/wiki/neural_network_hyperparameters.

[29] [Online]. Available: http://karpathy.github.io/2019/04/25/recipe/.

[30] [Online]. Available: https://www.kdnuggets.com/2019/11/designing-neural-networks.html.

[31] S. Kostadinov, "Recurrent Neural Networks with Python Quick Start Guide," 2018, p. 93.

[32] [Online]. Available: https://blog.dellemc.com/en-us/ai-deep-learning-unstructured-data-isilon/.

[33] [Online]. Available: https://www.statisticshowto.com/standardized-values-examples/.

[34] [Online]. Available: https://www.pyimagesearch.com/2019/10/14/why-is-my-validation-loss-lower-than-my-training-loss/.

[35] [Online]. Available: http://karpathy.github.io/2019/04/25/recipe/.

[36] [Online]. Available: https://www.kdnuggets.com/2019/11/designing-neural-networks.html.

[37] S. Theodoridis, "Machine Learning: A Bayesian and Optimization Perspective," AP, p. 923.

[38] J. Jordan, "Setting the learning rate of your neural network.," [Online]. Available: https://www.jeremyjordan.me/nn-learning-rate/.

[39] [Online]. Available: https://mlfromscratch.com/optimizers-explained/#/.

[40] [Online]. Available: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.

[41] J. L. B. Diederik P.Kingm, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION," https://arxiv.org/pdf/1412.6980.pdf, 2015.

[42] J. Brownlee, "Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions," 2020, pp. 245-251.

[43] [Online]. Available: https://medium.com/analytics-vidhya/regularisation-techniques-in-machine-learning-and-deep-learning-8102312e1ef3.

[44] [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html.

[45] Y. Bengio, "Practical Recommendations for Gradient-Based Training of Deep Architectures," arXiv report:1206.5533, 2012.

[46] [Online]. Available: https://www.geeksforgeeks.org/introduction-to-residual-networks/.

[47] [Online]. Available: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035.

[48] M. Clerc, Particle Swarm Optimization, ISTE, 2005.

[49] P. v. Laarhoven and E. Aarts, Simulated Annealing: Theory and Applications, Kluwer Academic Publishers, 1992.

[50] [Online]. Available: http://apmonitor.com/me575/index.php/Main/SimulatedAnnealing.

[51] S. Soltani-Mohammadi, M. Safa and H. Mokhtarib, "Comparison of particle swarm optimization and simulated annealing for locating additional boreholes considering combined variance minimization," *Computers & Geosciences,* vol. 95, pp. 146-155, October 2016.

[52] S. e. a. Chai, "A Comparison of Genetic Algorithm, Particle Swarm Optimization and Simulated Annealing in Real-Time Task Scheduling on CMP," *Advanced Materials Research,* vol. 679, pp. 77-81, 2013.

[53] M. Bagaram, "Comparison of Simulated Annealing and Particle SwarmOptimization on Reliability-Redundancy Problem," University Of Washington. 2, 2018.

APPENDIX A. FLOWSHEETS

Due to the images are so large, and some details might be not so visible, here are given the links to the different flowsheets.

GASIFIER FLOWSHEET

https://drive.google.com/file/d/1lNG2gHUA4RCBxSVWAbfu4APAIHUq4CZu/view?usp=sharing

CLEANING FLOWSHEET

https://drive.google.com/file/d/1s1Vne6L4VujFT7DZxgwBFjNcLC3OUmuZ/view?usp=sharing

WGS FLOWSHEET

https://drive.google.com/file/d/1iGHfOD2WKTo4n3P0heRGN6rlKKy6EwKC/view?usp=sharing

NH3 FLOWSHEET

https://drive.google.com/file/d/1iYE9xnWSAFTideuokdpmVIMWGq548rBY/view?usp=sharing

GLOBAL

https://drive.google.com/file/d/17NNOm18V8p5My2meGy3pbtVhMvmj0oJT/view?usp=sharing

APPENDIX B. VARIABLES SELECTION

GASIFIER

INPUT Streams (SLURRY-I, O2NC, RAD-BFW, CSC1BFW, CSC2BFW, GRAY-1, GRAY-2)

SLURRY_I (coal+water)

1. Mass flow ratio COAL/BIOMASS: This variable will allow to change the composition of the incoming fuel, which will have an effect on the amount of H2, CO and CO2 produced by the process. It is expected a reduction in the CO2 emissions when the biomass share is increased (reference). However the properties PROXANAL, ULTANAL AND SULFANAL of the both kind of fuels should be left constant (similar to Anthracite properties)

O2NC

2. Mass flow (total): This variable have a direct impact on the gasification reaction since the oxygen is a key reactant. By causing a variation in the ratio of mole C/mole O2 it is expected to affect the proportion of H2, CO and CO2 generated (but mainly the last two).

RAD-BFW, CSC1BFW, CSC2BFW

3. Mass flow of the HP steam streams (Optional): This mass flow controls the main stream outlet temperature in the 3 heat exchangers that operate in the gasifier.

OUTPUT Streams: (TOSUMP, HPS-OUT2, BLACKWTR, SYNGAS-B)

CLEANING BLOCK

INPUT Streams (GAS-A (Syngas), N2-A, LEAN-1)

LEAN-1: (MeOH (99.9%) +H2S)

4. Mole flow (total): This stream of methanol is used as an absorbent to remove the CO2 and H2S from the syngas. By increasing this flow it is expected further removal of CO2 and H2S.

N2-A:

5. Mole Flow (total): This stream is used not only to remove the H2 from the final H2S stream but also as a reactant in the ammonia production.

COLUMN unit (H2SSTR: separates H2S from MeOH)

6. Distillate rate (Optional): By increasing the distillate rate it is expected to increase the H2S concentration in the final. Although the amount of H2S obtained is not so high compared with the syngas main flow.

SPLITTER (CO2RICH)

7. Split Proportion (Optional): splits the CO2RICH stream that comes from the CO2ABS unit. The small fraction (13%) is recycled back to the H2S absorption unit and the rest goes to CO2 separation. There exists an equilibrium point around the default value to get the optimum separation of H2S.

OUTPUT Streams: SYNGAS (treated), COND1, H2S, LEAN, CO2, LEAN-0B

WGS BLOCK

INPUT Streams: (SYNGAS SSTEAM, LEAN)

LEAN

8. Temperature (optional): The lean stream temperature affects the absorption of CO2 in the absorber.
9. Mole flow (total): Controls the absorption of CO2. The more flow the more absorption

SSTEAM

10. Mole flow (optional): Although it is an important variable this quantity right now is set equal as the syngas mole flow by means of a calculator. It is expected that when it is increased the steam flow, H2 and CO2 increased also until all CO reacts. However, changes in this variable leads to many errors in the collection of data.

OUTPUT Streams: (H2, OFF, LEAN-1, LIQ3)

NH3 BLOCK

INPUT Streams: (RXFEED, ABSWTR)

ABSWTR: (water used in the NH3ABS unit)

11. Mole Flow (optional): This stream is a water flow that absorbs NH3 in a second stage. The more water flow the more NH3 separated at this point.

HEATER (FHDTX)

12. Temperature and pressure (optional): High pressures and temperatures are a characteristic of the Haber-Bosch process. Those should be left as default they obey to standard process technique, modify these variables results unpractical in economical and technical terms.

SPLITTERS (PRIMSPLT, SECSPLT, RCYCSPLT): The proportion will determine the efficiency of the NH3 formation in the bed reactors.

13. Split proportion PRIMSPLT: Splits the flow (86.1%) to be heated by the 3 heat exchangers (COOLG stream), and (13.9%) to the second splitter (MAIN stream).
14. Split proportion SECSPLT: Splits the flow (20%) for Bed1 (FD1-A stream), and (80%) for bed2 (FD1-B stream).
15. Split proportion RCYCSPLT: Splits the flow (10%) to be recycled.

REACTORS (BED1, BED2, BED3 outlet temperature) Determine the kinetic of the reaction

16. Temperature BED1
17. Temperature BED2
18. Temperature BED3

COOLER (CHLLR: cool down NH3 stream)

19. Temperature (Optional): Decreases the temperature from 454 to 0°C to make the separation of the NH3 in the liquid phase.

OUTPUT Streams: HICO, LIQ1, AQ2, AQ3, VENT

METHANIZATION BLOCK

INPUT Streams (SYNGAS, H2)

METHANIZER

20. Temperature (optional): Sets the temperature of the final methane stream. Depending on this value is the amount of energy generated by the methanizer.

OUTPUT Streams (HICH4)

SYNGAS SPLITTER (SYNSPLIT)

SYNSPLIT This split (power/methane/ammonia) allows to make the plant flexible when market variations occur.

21. Split proportions for WGS.
22. Split proportions for Methanization.

Quantities to collect

Energy Streams on each block

(It is needed also to present a table with the different parameters (T, P, composition) of the Input Streams for each block)

APPENDIX C. ACTIVATION AND LOSS FUNCTIONS FORMULATION

LOSS FUNCTIONS

MSE

mse corresponds with the sum of squared difference between the expected and predicted values.

$$mean\ squared\ error = mse = \frac{\sum_{i=1}^{m}\left(y_i^{pred} - y_i^{exp}\right)^2}{m}$$

MAE

mae corresponds with the sum of the absolute differences between the expected and predicted values. It measures the average magnitude of errors in a set of predictions, without considering their directions.

$$mean\ absolute\ error = mse = \frac{\sum_{i=1}^{m}\left|y_i^{pred} - y_i^{exp}\right|}{m}$$

HUBER LOSS

It is a function that corresponds to the absolute error when the parameter $\delta \rightarrow \infty$, and takes the form of the mse function when the parameter $\delta \rightarrow 0$.

$$Huber = \begin{cases} \frac{1}{2}\sum_{i=1}^{m}\left(y_i^{pred} - y_i^{exp}\right)^2 & for\ \left|y_i^{pred} - y_i^{exp}\right| \leq \delta \\ \delta \sum_{i=1}^{m}\left|y_i^{pred} - y_i^{exp}\right| - \frac{1}{2}\delta^2 & otherwise \end{cases}$$

LOGCOSH

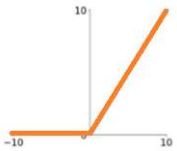
Log-cosh corresponds to the logarithm of the hyperbolic cosine of the error (difference between the predicted and expected values).

$$Logcosh = \sum_{i=1}^{m} log\left(cosh\left(y_i^{pred} - y_i^{exp}\right)\right)$$

ACTIVATION FUNCTIONS

RELU

ReLU it is known as the Rectified Linear Unit. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time and is computational very efficient due to its linear term for $x > 0$.

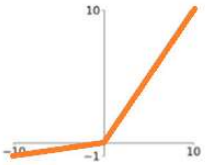


ReLU

$$\max(0, x)$$

LRELU

Leaky ReLU is a variation of ReLU. It is not strictly zero for values x<0 but instead it has a small linear variation with negative slope. It is intended to address the problem of the ReLU function which deactivate the neurons in the region for $x < 0$.

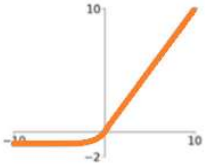


Leaky ReLU

$$\max(0.1x, x)$$

ELU (Exponential Linear Unit)

This is a variant of ReLU. The function is not zero for $x < 0$ but instead it takes a negative value with an exponential transition.



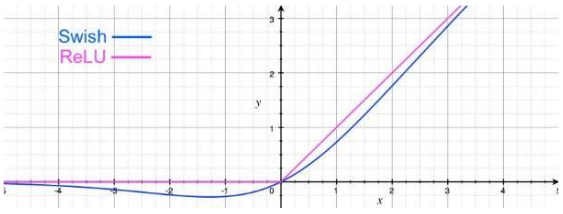
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

SWISH

Swish is a not so known activation function which was developed by researchers at Google. Swish is as computationally efficient activation function as ReLU and have shown better performance than ReLU on deep network models. It is defined by the next expression.

$$f(x) = x * sigmoid(x) = \frac{x}{1 - e^{-x}}$$



APPENDIX D. MINI MODEL COMPLEMENTARY RESULTS

1) 2-step study: Dimensional analysis to get the operative ranges of the variables

One dimensional study:

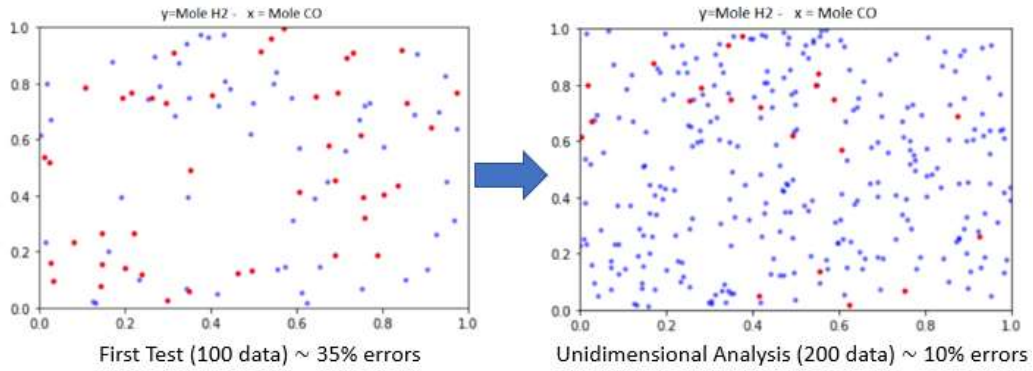


Figure Ap 1. One dimensional results example error reduction.

Two-dimensional study: Ranges of less errors were selected upon the unidimensional results but taking care to not shrink the investigated space too much:

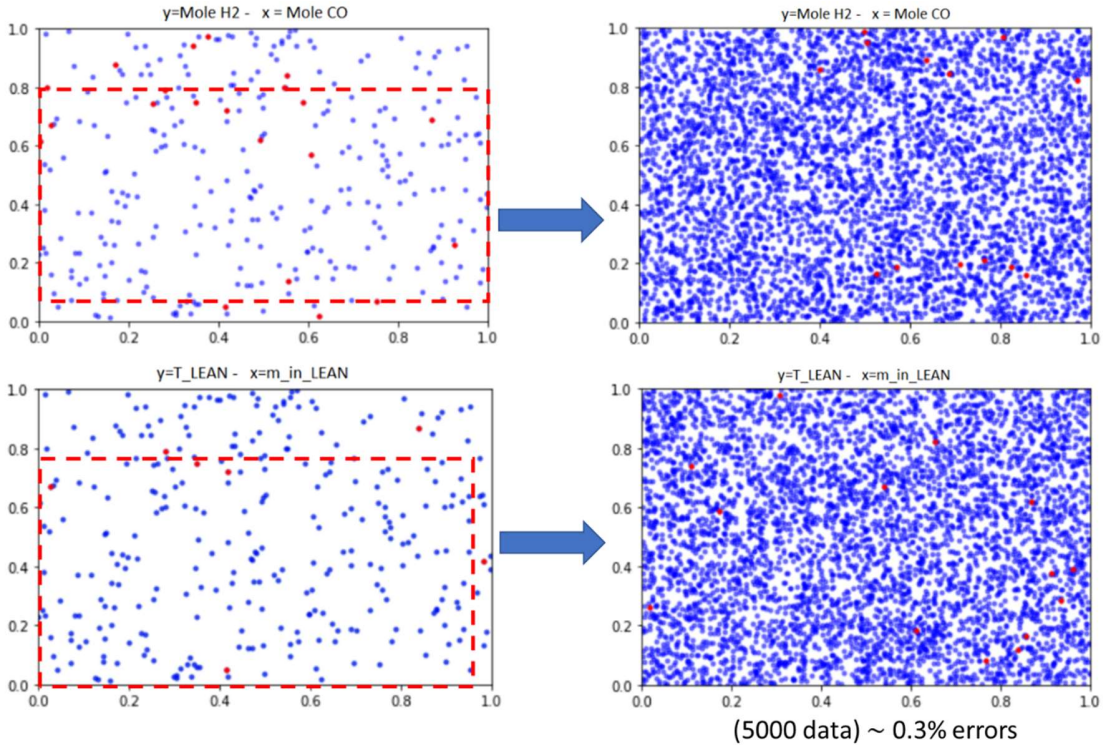


Figure Ap 2. Two-dimensional analysis. Shrinking the search space to reduce errors

The reduction in the number of errors is absolutely significant. Therefore, the 2-step study is useful and necessary to reduce the total number of convergence errors.

2) Sorting method:

Sorting Method Results:

The three sorting methods were evaluated upon the same set. Two sets sizes were evaluated: 5000 and 10000. Both generated with MC as the random generator.

Table Ap 1. Sorting methods results. Convergence errors comparison.

	No. Errors		%dif	
Method	**5000 spl**	**10000 spl**	**5000 spl**	**10000 spl**
None	10	23	-	-
HCS	10	23	0.0	0.0
TSPH	7	17	-30.0	-26.1

As it can be seen, the TSPH method had the less amount of Aspen convergence errors whereas HCS had the same amount of errors than when no method was applied.

3) Random Generator Results:

The three random generators were used to create different set sizes (10000 samples and 20000 samples), and the results in terms of convergence errors and total simulation time are given in Table Ap 2.

Table Ap 2. Results of the convergence errors and simulation time. for the three random generators tested.

	No. Errors		Time(sec)		%dif	
Method	**10000 spl**	**20000 spl**	**10000 spl**	**20000 spl**	**10000 spl**	**20000 spl**
MCS+TSPH	17	55	26097	55050	-	-
LHS+TSPH	21	45	26585	56166	1.9	2.0
QMCS+TSPH	15	48	24869	53895	-4.7	-2.1

As seen in Table Ap 2, there is no significant difference in terms of number of convergence errors for the three methods. So, all methods behave more or less the same in this respect. However, in the total simulation time, the QMCS method presented a slightly lower time compared to the MCS result.

In Figure Ap 3, the data distribution generated with the different random methods tested it is presented. A red dot corresponds to a convergence error sample set. Below each plot can be found the number of the variable according with Table 4 that defines the x and y axis (x, y).

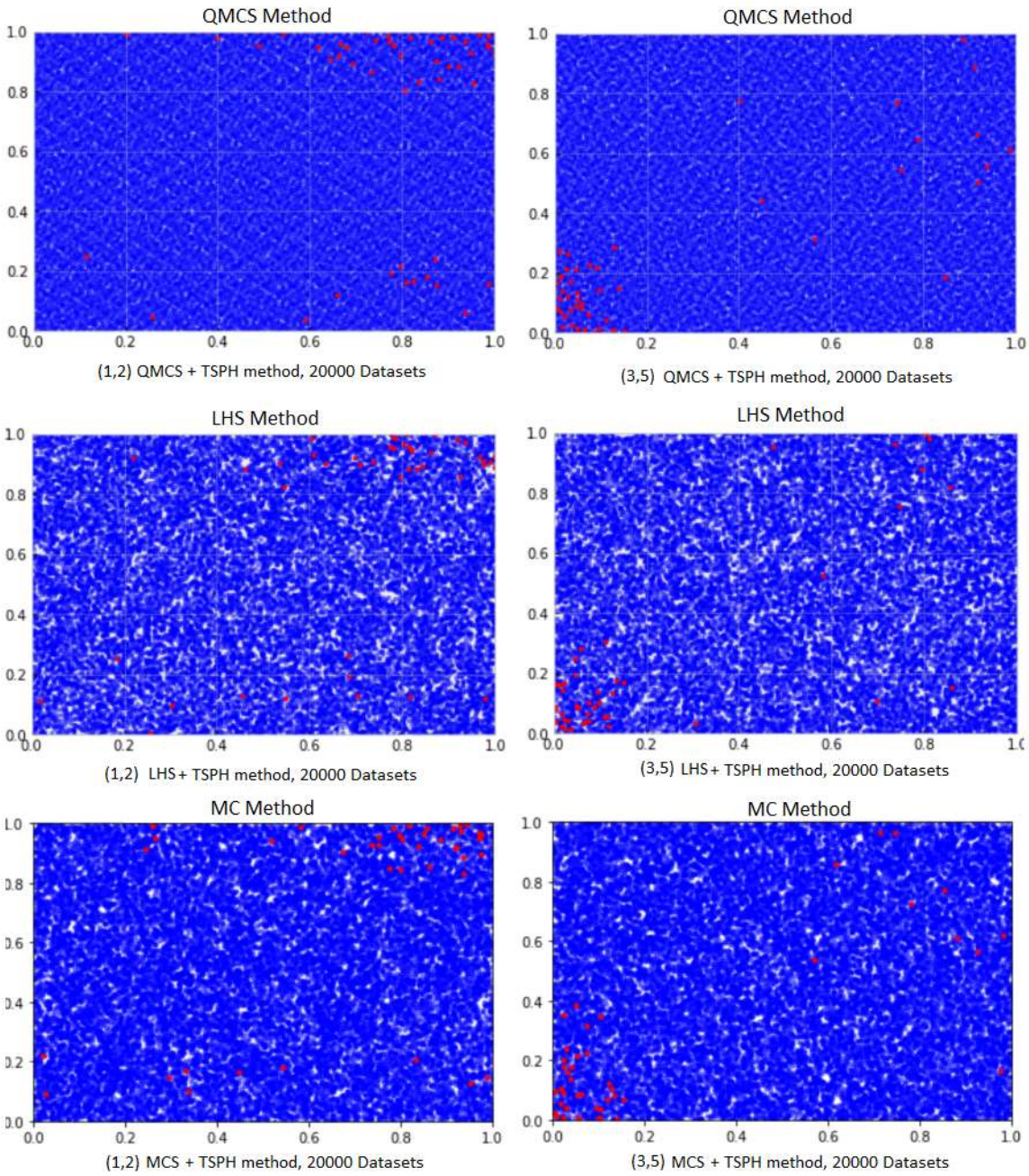


Figure Ap 3. Data distribution for the three random generators used. Red dots are the convergence errors.

As can be seen in Figure Ap 3, the QMCS method filled the entire space more homogenously than the other methods which presented many "white" spaces for the same number of samples (20000).

Quick Neural Network Evaluation

Also a quick Neural Network implementation was made with a 3L_[100,200,300]_1200e_1024b (see Section 4.1 to understand the notation), Adam optimizer (LR:1e-4), *mse* as loss function and *mae* as the metric. Three sets of 5000 samples were generated for each random generator (so 9 sets in total). Then, the three sets were evaluated independently in the network; the train/crossv ratio was (10:1). The results of the loss function, the metric and the sum of the relative errors for the Crossvalidation set of each test are given in the next table:

Table Ap 3. Results of Loss function, mae, and sum of error in the Crossv set for the different tst made.

		mse Crossv		**mae Crossv**		**Sum Error Crossv**	
Method	**Set**	**Result**	**Average**	**Result**	**Average**	**Result**	**Average**
MC+TSPH	Set 1	0.0087	**0.0083**	0.0157	**0.0152**	1314	**1056**
	Set 2	0.0079		0.0152		766	
	Set 3	0.0083		0.0147		1088	
LHS+TSPH	Set 1	0.0111	**0.0099**	0.0161	**0.0157**	1974	**1125**
	Set 2	0.0087		0.0155		577	
	Set 3	0.0099		0.0155		822	
QMCS+TSPH	Set 1	0.0082	**0.0081**	0.0145	**0.0145**	1047	**792**
	Set 2	0.0078		0.0141		581	
	Set 3	0.0082		0.0149		748	

As the table shows, the method with the lowest average value for the 3 quantities measured (mse, mae and sum error) was **QMCS+TSPH**. Probably the characteristic of homogeneity provided by the QMCS method allows the network to better describe the space and make it more robust against an input set change (Crossv in this case).

4) Results Evaluation. Actual values

In Table Ap 4, a comparison is given between the predicted results of the Mini model neural network (Ypred) and the expected values (Yexp), for 3 samples picked randomly. The amounts presented correspond to the "actual" values, i.e. after inverting the standardization transformation. Temperature in °C, Pressure in bar, Enthalpy in Gcal/hr, the chemical components were taken in kmol/hr.

Table Ap 4. Comparison between predicted and expected "actual" values, for 3 samples.

	Sample 3536		Sample 838		Sample 3154			Sample 3536		Sample 838		Sample 3154	
Variable	**Ypred**	**Yexp**	**Ypred**	**Yexp**	**Ypred**	**Yexp**	**Variable**	**Ypred**	**Yexp**	**Ypred**	**Yexp**	**Ypred**	**Yexp**
'H2_T_out'	18.13	17.93	16.79	16.65	11.20	11.51	'LEAN-1_T_out'	89.63	89.66	87.57	87.08	84.86	84.80
'H2_P_out'	24.13	24.13	24.13	24.13	24.13	24.13	'LEAN-1_P_out'	6.89	6.89	6.89	6.89	6.89	6.89
'H2_H_out'	-46.00	-45.83	-45.92	-45.94	-43.44	-43.47	'LEAN-1_H_out'	-175.83	-175.84	-193.74	-193.74	-197.54	-196.71
'H2_N2'	782.42	785.62	547.82	542.26	794.60	794.58	'LEAN-1_N2'	0.11	0.11	0.09	0.09	0.14	0.14
'H2_O2'	4.66	4.66	4.64	4.64	4.65	4.65	'LEAN-1_O2'	0.00	0.00	0.00	0.00	0.00	0.00
'H2_AR'	17.85	17.85	17.80	17.80	17.81	17.81	'LEAN-1_AR'	0.00	0.00	0.01	0.01	0.01	0.01
'H2_H2O'	0.00	0.00	0.00	0.00	0.00	0.00	'LEAN-1_H2O'	5.07	3.83	3.80	3.14	2.49	2.32
'H2_CO'	135.78	135.46	145.34	145.04	121.07	120.98	'LEAN-1_CO'	0.00	0.00	0.00	0.00	0.00	0.00
'H2_CO2'	448.01	446.56	444.74	444.99	423.78	424.26	'LEAN-1_CO2'	56.33	56.46	66.81	66.88	72.10	71.44
'H2_COS'	0.00	0.00	0.00	0.00	0.00	0.00	'LEAN-1_COS'	0.00	0.00	0.00	0.00	0.00	0.00
'H2_H2S'	0.00	0.00	0.00	0.00	0.00	0.00	'LEAN-1_H2S'	0.03	0.03	0.03	0.03	0.03	0.03
'H2_H2'	1475.38	1476.94	1422.59	1425.97	1334.05	1334.47	'LEAN-1_H2'	0.00	0.00	0.01	0.01	0.00	0.00
'H2_CH4'	0.06	0.06	0.06	0.06	0.06	0.06	'LEAN-1_CH4'	0.00	0.00	0.00	0.00	0.00	0.00
'H2_MEOH'	0.57	0.57	0.70	0.70	0.66	0.65	'LEAN-1_MEOH'	3035.93	3043.94	3350.62	3342.45	3395.75	3384.70
'OFF_T_out'	89.70	89.66	87.09	87.08	84.95	84.80	'LIQ3_T_out'	24.41	23.95	21.25	21.62	16.02	16.33
'OFF_P_out'	6.89	6.89	6.89	6.89	6.89	6.89	'LIQ3_P_out'	24.16	24.13	24.14	24.13	24.20	24.13
'OFF_H_out'	-56.25	-56.65	-64.14	-64.17	-57.79	-58.22	'LIQ3_H_out'	-9.09	-9.08	-9.85	-9.81	-8.23	-8.21
'OFF_N2'	7.35	7.39	6.15	6.19	8.65	8.59	'LIQ3_N2'	0.00	0.00	0.00	0.00	0.00	0.00
'OFF_O2'	0.07	0.07	0.09	0.09	0.08	0.08	'LIQ3_O2'	0.00	0.00	0.00	0.00	0.00	0.00
'OFF_AR'	0.25	0.25	0.30	0.30	0.29	0.29	'LIQ3_AR'	0.00	0.00	0.00	0.00	0.00	0.00
'OFF_H2O'	0.00	0.16	0.08	0.12	0.03	0.07	'LIQ3_H2O'	132.43	131.89	142.55	142.36	119.30	119.03
'OFF_CO'	0.42	0.42	0.57	0.57	0.44	0.44	'LIQ3_CO'	0.00	0.00	0.00	0.00	0.00	0.00
'OFF_CO2'	450.55	452.34	530.07	530.06	492.25	492.82	'LIQ3_CO2'	0.01	0.01	0.01	0.01	0.01	0.01
'OFF_COS'	0.00	0.00	0.00	0.00	0.00	0.00	'LIQ3_COS'	0.00	0.00	0.00	0.00	0.00	0.00
'OFF_H2S'	0.06	0.06	0.06	0.06	0.05	0.05	'LIQ3_H2S'	0.00	0.00	0.00	0.00	0.00	0.00
'OFF_H2'	1.20	1.21	1.50	1.51	1.29	1.30	'LIQ3_H2'	0.00	0.00	0.00	0.00	0.00	0.00
'OFF_CH4'	0.00	0.00	0.00	0.00	0.00	0.00	'LIQ3_CH4'	0.00	0.00	0.00	0.00	0.00	0.00
'OFF_MEOH'	304.80	301.52	305.90	306.42	255.65	254.35	'LIQ3_MEOH']	0.10	0.10	0.12	0.12	0.12	0.12

APPENDIX E. FULL IGCC MODEL COMPLEMENTARY RESULTS

1) Table with the streams and attributes read on each of them for the Full IGCC model

Input set

Table Ap 5. Input streams and its attributes to Read

Block	Stream	Attributes to read	N atrib
GASFR	SLURRY-I	T,P,H,H2O,COAL,BIOMASS,COALASH	7
	OXYGEN	T,P,H,N2,O2,AR	6
	RAD-BFW	T,P,H,H2O	4
	CSC1BFW	T,P,H,H2O	4
	CSC2BFW	T,P,H,H2O	4
	GRAY-1	T,P,H,H2O,H3N,NH4+,H3O+,OH-	8
	GRAY-2	T,P,H,H2O,H3N,NH4+,H3O+,OH-	8
	HEATIN	Heat	1
CLEANING	GAS-A	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	N2-IN	T,P,H,N2,O2,AR	6
	LEAN-1	T,P,H,H2S,MEOH	5
	HEATIN	Heat	1
WGS	SYNGAS	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	SSTEAM	T,P,H,H2O	4
	LEAN	T,P,H,CO2,MEOH	5
	HEATIN	Heat	1
NH3	RXFEED	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	ABSWTR	T,P,H,H2O,H3O+,OH-	6
	HEATIN	Heat	1
METHANIZ	H2	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	CO	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	HEATIN	Heat	1
			147

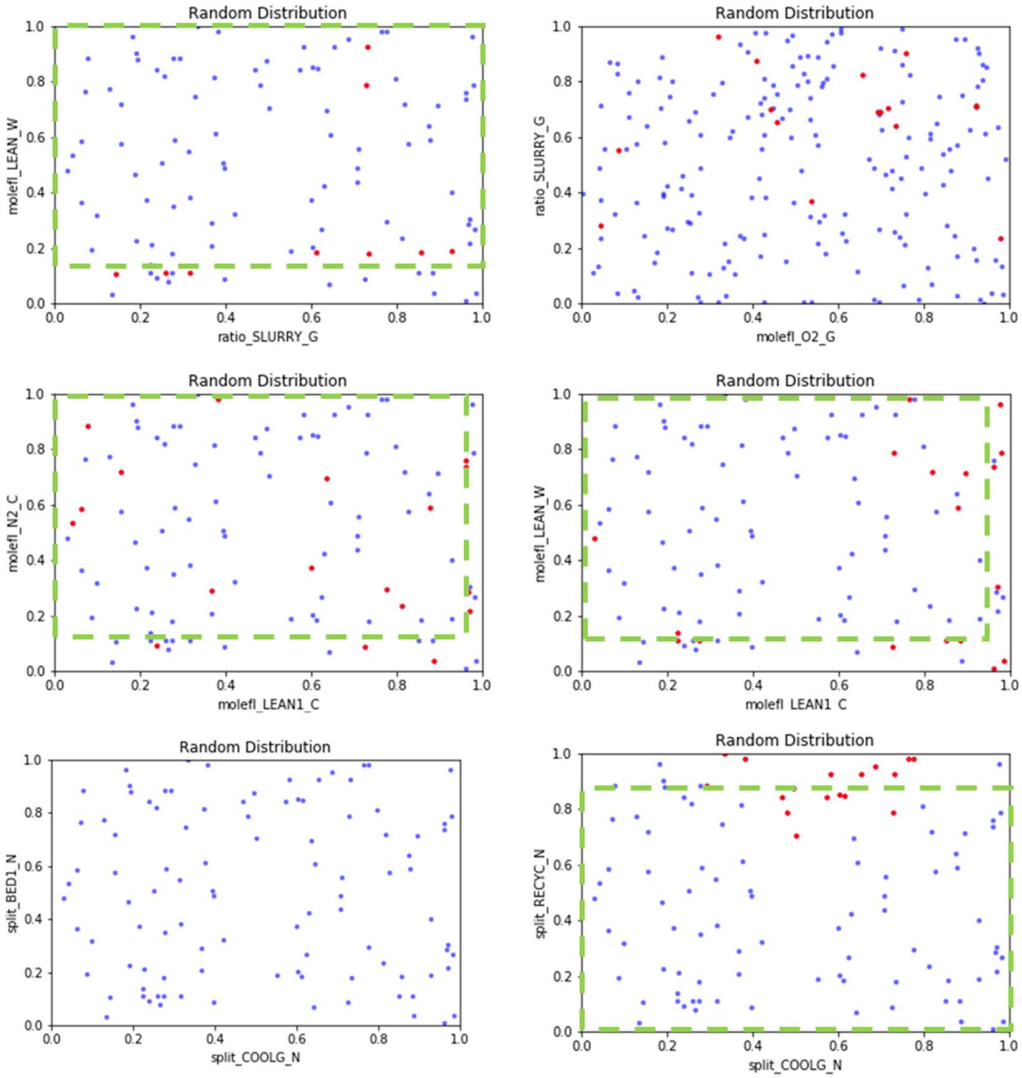
Output Set

Table Ap 6. Output streams and its attributes to read.

Block	Stream	Attributes to read	Nparam
GASFR	SYNGAS-B	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,COAL,BIOMASS,COALASH,OTHERS	18
	HPS-OUT2	T,P,H,H2O,OTHERS	5
	BLACKWTR	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,COAL,BIOMASS,COALASH,OTHERS	18
	TOSUMP	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,COAL,BIOMASS,COALASH,OTHERS	18
	HEATOUT	HEAT	1
CLEANING	TREATED	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	COND1	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	RECYC-3	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	H2S	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	LEAN	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	CO2	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	LEAN-0B	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	CONDENS	HEATOUT	1
	HEATOUT	HEAT	1
WGS	H2	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	OFF	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	LEAN-1	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	LIQ3	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	HEATOUT	HEAT	1
NH3	HICO	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	LIQ1	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	AQ2	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	AQ3	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	VENT	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	HEATOUT	HEAT	1
METHANIZ	HICH4	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
	HEATOUT	HEAT	1
GLOBAL	SGB	T,P,H,N2,O2,H2O,CO,CO2,COS,H3N,H2S,H2,CH4,MEOH,OTHERS	15
			335

2) Bidimensional Analysis

Below are shown the different graphs obtained in the bidimensional analysis performed over the variables chosen for the Full IGCC Aspen model. To describe the plant better, it was desired to keep the input space as large as possible. Therefore, some of the variables preserved its one-dimensional ranges because no significant error regions were found, or because they were too spread out in the space that it was not possible to define a range. After the two-dimensional study was done the amount of convergence errors was reduced from 70% (with the first ranges) to 40% of the total amount of simulations.



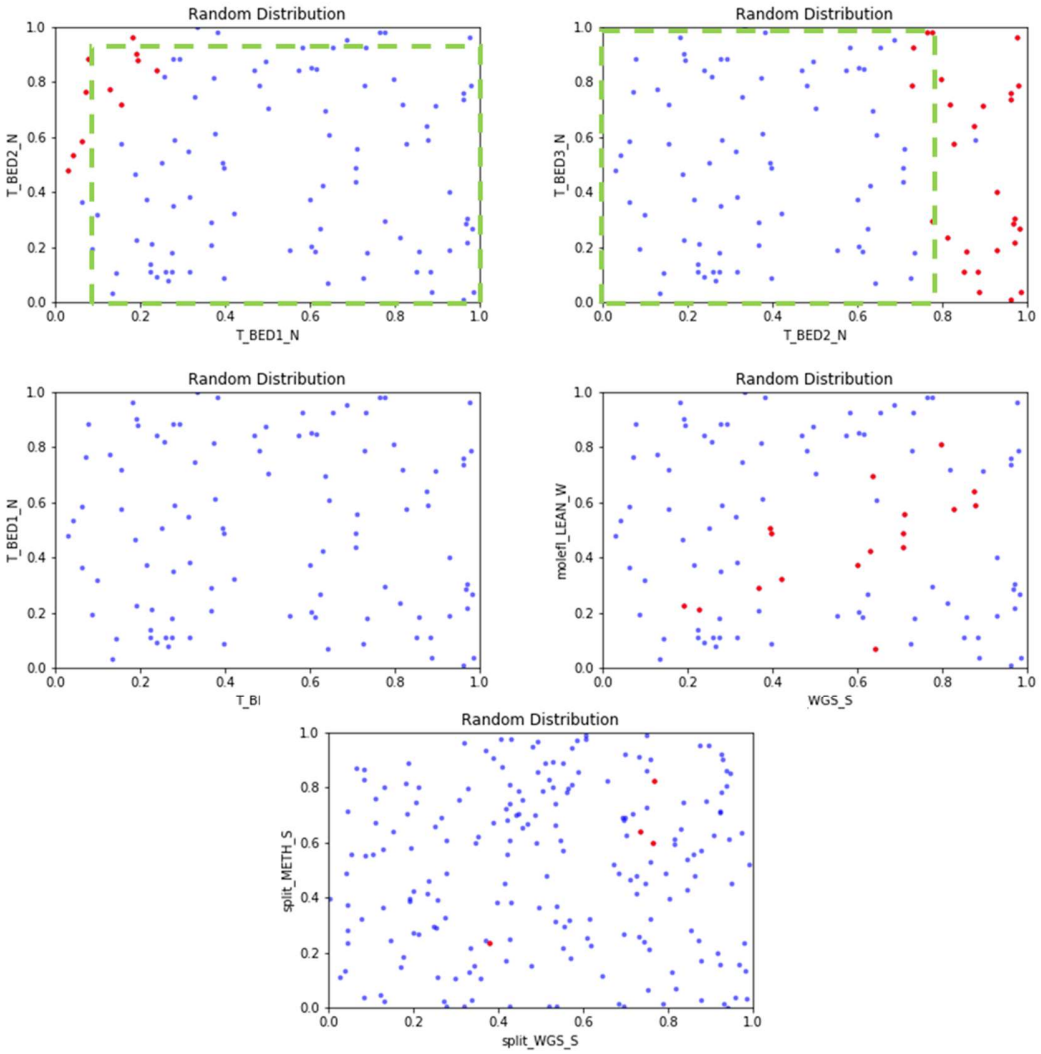


Figure Ap 4. Bidimensional study graphs for the chosen variables. Full IGCC model.

3) Table with output variable and number correspondence

Table Ap 7. Number and Output Variable Correspondence

n	Variable	n	Variable	n	Variable	n	Variable
1	GASFR_SYNGAS-B_T_out	51	GASFR_TOSUMP_H3N	101	CLEANING_RECYC-3_H2S	151	CLEANING_LEAN-0B_T_ou
2	GASFR_SYNGAS-B_P_out	52	GASFR_TOSUMP_H2S	102	CLEANING_RECYC-3_H2	152	CLEANING_LEAN-0B_P_ou
3	GASFR_SYNGAS-B_H_out	53	GASFR_TOSUMP_H2	103	CLEANING_RECYC-3_CH4	153	CLEANING_LEAN-0B_H_ou
4	GASFR_SYNGAS-B_N2	54	GASFR_TOSUMP_CH4	104	CLEANING_RECYC-3_MEOH	154	CLEANING_LEAN-0B_N2
5	GASFR_SYNGAS-B_O2	55	GASFR_TOSUMP_MEOH	105	CLEANING_RECYC-3_OTHE	155	CLEANING_LEAN-0B_O2
6	GASFR_SYNGAS-B_H2O	56	GASFR_TOSUMP_COAL	106	CLEANING_H2S_T_out	156	CLEANING_LEAN-0B_H2O
7	GASFR_SYNGAS-B_CO	57	GASFR_TOSUMP_BIOMASS	107	CLEANING_H2S_P_out	157	CLEANING_LEAN-0B_CO
8	GASFR_SYNGAS-B_CO2	58	GASFR_TOSUMP_COALASH	108	CLEANING_H2S_H_out	158	CLEANING_LEAN-0B_CO2
9	GASFR_SYNGAS-B_COS	59	GASFR_TOSUMP_OTHERS	109	CLEANING_H2S_N2	159	CLEANING_LEAN-0B_COS
10	GASFR_SYNGAS-B_H3N	60	GASFR_HEATOUT	110	CLEANING_H2S_O2	160	CLEANING_LEAN-0B_H3N
11	GASFR_SYNGAS-B_H2S	61	CLEANING_TREATED_T_ou	111	CLEANING_H2S_H2O	161	CLEANING_LEAN-0B_H2S
12	GASFR_SYNGAS-B_H2	62	CLEANING_TREATED_P_ou	112	CLEANING_H2S_CO	162	CLEANING_LEAN-0B_H2
13	GASFR_SYNGAS-B_CH4	63	CLEANING_TREATED_H_ou	113	CLEANING_H2S_CO2	163	CLEANING_LEAN-0B_CH4
14	GASFR_SYNGAS-B_MEOH	64	CLEANING_TREATED_N2	114	CLEANING_H2S_COS	164	CLEANING_LEAN-0B_MEO
15	GASFR_SYNGAS-B_COAL	65	CLEANING_TREATED_O2	115	CLEANING_H2S_H3N	165	CLEANING_LEAN-0B_OTHE
16	GASFR_SYNGAS-B_BIOMAS	66	CLEANING_TREATED_H2O	116	CLEANING_H2S_H2S	166	CLEANING_CONDENS_HEA
17	GASFR_SYNGAS-B_COALAS	67	CLEANING_TREATED_CO	117	CLEANING_H2S_H2	167	CLEANING_HEATOUT
18	GASFR_SYNGAS-B_OTHERS	68	CLEANING_TREATED_CO2	118	CLEANING_H2S_CH4	168	WGS_H2_T_out
19	GASFR_HPS-OUT2_T_out	69	CLEANING_TREATED_COS	119	CLEANING_H2S_MEOH	169	WGS_H2_P_out
20	GASFR_HPS-OUT2_P_out	70	CLEANING_TREATED_H3N	120	CLEANING_H2S_OTHERS	170	WGS_H2_H_out
21	GASFR_HPS-OUT2_H_out	71	CLEANING_TREATED_H2S	121	CLEANING_LEAN_T_out	171	WGS_H2_N2
22	GASFR_HPS-OUT2_H2O	72	CLEANING_TREATED_H2	122	CLEANING_LEAN_P_out	172	WGS_H2_O2
23	GASFR_HPS-OUT2_OTHERS	73	CLEANING_TREATED_CH4	123	CLEANING_LEAN_H_out	173	WGS_H2_H2O
24	GASFR_BLACKWTR_T_out	74	CLEANING_TREATED_MEO	124	CLEANING_LEAN_N2	174	WGS_H2_CO
25	GASFR_BLACKWTR_P_out	75	CLEANING_TREATED_OTHE	125	CLEANING_LEAN_O2	175	WGS_H2_CO2
26	GASFR_BLACKWTR_H_out	76	CLEANING_COND1_T_out	126	CLEANING_LEAN_H2O	176	WGS_H2_COS
27	GASFR_BLACKWTR_N2	77	CLEANING_COND1_P_out	127	CLEANING_LEAN_CO	177	WGS_H2_H3N
28	GASFR_BLACKWTR_O2	78	CLEANING_COND1_H_out	128	CLEANING_LEAN_CO2	178	WGS_H2_H2S
29	GASFR_BLACKWTR_H2O	79	CLEANING_COND1_N2	129	CLEANING_LEAN_COS	179	WGS_H2_H2
30	GASFR_BLACKWTR_CO	80	CLEANING_COND1_O2	130	CLEANING_LEAN_H3N	180	WGS_H2_CH4
31	GASFR_BLACKWTR_CO2	81	CLEANING_COND1_H2O	131	CLEANING_LEAN_H2S	181	WGS_H2_MEOH
32	GASFR_BLACKWTR_COS	82	CLEANING_COND1_CO	132	CLEANING_LEAN_H2	182	WGS_H2_OTHERS
33	GASFR_BLACKWTR_H3N	83	CLEANING_COND1_CO2	133	CLEANING_LEAN_CH4	183	WGS_OFF_T_out
34	GASFR_BLACKWTR_H2S	84	CLEANING_COND1_COS	134	CLEANING_LEAN_MEOH	184	WGS_OFF_P_out
35	GASFR_BLACKWTR_H2	85	CLEANING_COND1_H3N	135	CLEANING_LEAN_OTHERS	185	WGS_OFF_H_out
36	GASFR_BLACKWTR_CH4	86	CLEANING_COND1_H2S	136	CLEANING_CO2_T_out	186	WGS_OFF_N2
37	GASFR_BLACKWTR_MEOH	87	CLEANING_COND1_H2	137	CLEANING_CO2_P_out	187	WGS_OFF_O2
38	GASFR_BLACKWTR_COAL	88	CLEANING_COND1_CH4	138	CLEANING_CO2_H_out	188	WGS_OFF_H2O
39	GASFR_BLACKWTR_BIOMA	89	CLEANING_COND1_MEOH	139	CLEANING_CO2_N2	189	WGS_OFF_CO
40	GASFR_BLACKWTR_COALA	90	CLEANING_COND1_OTHER	140	CLEANING_CO2_O2	190	WGS_OFF_CO2
41	GASFR_BLACKWTR_OTHER	91	CLEANING_RECYC-3_T_out	141	CLEANING_CO2_H2O	191	WGS_OFF_COS
42	GASFR_TOSUMP_T_out	92	CLEANING_RECYC-3_P_ou	142	CLEANING_CO2_CO	192	WGS_OFF_H3N
43	GASFR_TOSUMP_P_out	93	CLEANING_RECYC-3_H_ou	143	CLEANING_CO2_CO2	193	WGS_OFF_H2S
44	GASFR_TOSUMP_H_out	94	CLEANING_RECYC-3_N2	144	CLEANING_CO2_COS	194	WGS_OFF_H2
45	GASFR_TOSUMP_N2	95	CLEANING_RECYC-3_O2	145	CLEANING_CO2_H3N	195	WGS_OFF_CH4
46	GASFR_TOSUMP_O2	96	CLEANING_RECYC-3_H2O	146	CLEANING_CO2_H2S	196	WGS_OFF_MEOH
47	GASFR_TOSUMP_H2O	97	CLEANING_RECYC-3_CO	147	CLEANING_CO2_H2	197	WGS_OFF_OTHERS
48	GASFR_TOSUMP_CO	98	CLEANING_RECYC-3_CO2	148	CLEANING_CO2_CH4	198	WGS_LEAN-1_T_out
49	GASFR_TOSUMP_CO2	99	CLEANING_RECYC-3_COS	149	CLEANING_CO2_MEOH	199	WGS_LEAN-1_P_out
50	GASFR_TOSUMP_COS	100	CLEANING_RECYC-3_H3N	150	CLEANING_CO2_OTHERS	200	WGS_LEAN-1_H_out

n	Variable	n	Variable	n	Variable
201	WGS_LEAN-1_N2	251	NH3_LIQ1_CO2	301	NH3_VENT_CH4
202	WGS_LEAN-1_O2	252	NH3_LIQ1_COS	302	NH3_VENT_MEOH
203	WGS_LEAN-1_H2O	253	NH3_LIQ1_H3N	303	NH3_VENT_OTHERS
204	WGS_LEAN-1_CO	254	NH3_LIQ1_H2S	304	NH3_HEATOUT
205	WGS_LEAN-1_CO2	255	NH3_LIQ1_H2	305	METHANIZ_HICH4_T_out
206	WGS_LEAN-1_COS	256	NH3_LIQ1_CH4	306	METHANIZ_HICH4_P_out
207	WGS_LEAN-1_H3N	257	NH3_LIQ1_MEOH	307	METHANIZ_HICH4_H_out
208	WGS_LEAN-1_H2S	258	NH3_LIQ1_OTHERS	308	METHANIZ_HICH4_N2
209	WGS_LEAN-1_H2	259	NH3_AQ2_T_out	309	METHANIZ_HICH4_O2
210	WGS_LEAN-1_CH4	260	NH3_AQ2_P_out	310	METHANIZ_HICH4_H2O
211	WGS_LEAN-1_MEOH	261	NH3_AQ2_H_out	311	METHANIZ_HICH4_CO
212	WGS_LEAN-1_OTHERS	262	NH3_AQ2_N2	312	METHANIZ_HICH4_CO2
213	WGS_LIQ3_T_out	263	NH3_AQ2_O2	313	METHANIZ_HICH4_COS
214	WGS_LIQ3_P_out	264	NH3_AQ2_H2O	314	METHANIZ_HICH4_H3N
215	WGS_LIQ3_H_out	265	NH3_AQ2_CO	315	METHANIZ_HICH4_H2S
216	WGS_LIQ3_N2	266	NH3_AQ2_CO2	316	METHANIZ_HICH4_H2
217	WGS_LIQ3_O2	267	NH3_AQ2_COS	317	METHANIZ_HICH4_CH4
218	WGS_LIQ3_H2O	268	NH3_AQ2_H3N	318	METHANIZ_HICH4_MEOH
219	WGS_LIQ3_CO	269	NH3_AQ2_H2S	319	METHANIZ_HICH4_OTHERS
220	WGS_LIQ3_CO2	270	NH3_AQ2_H2	320	METHANIZ_HEATOUT
221	WGS_LIQ3_COS	271	NH3_AQ2_CH4	321	SGB_T_out
222	WGS_LIQ3_H3N	272	NH3_AQ2_MEOH	322	SGB_P_out
223	WGS_LIQ3_H2S	273	NH3_AQ2_OTHERS	323	SGB_H_out
224	WGS_LIQ3_H2	274	NH3_AQ3_T_out	324	SGB_N2
225	WGS_LIQ3_CH4	275	NH3_AQ3_P_out	325	SGB_O2
226	WGS_LIQ3_MEOH	276	NH3_AQ3_H_out	326	SGB_H2O
227	WGS_LIQ3_OTHERS	277	NH3_AQ3_N2	327	SGB_CO
228	WGS_HEATOUT	278	NH3_AQ3_O2	328	SGB_CO2
229	NH3_HICO_T_out	279	NH3_AQ3_H2O	329	SGB_COS
230	NH3_HICO_P_out	280	NH3_AQ3_CO	330	SGB_H3N
231	NH3_HICO_H_out	281	NH3_AQ3_CO2	331	SGB_H2S
232	NH3_HICO_N2	282	NH3_AQ3_COS	332	SGB_H2
233	NH3_HICO_O2	283	NH3_AQ3_H3N	333	SGB_CH4
234	NH3_HICO_H2O	284	NH3_AQ3_H2S	334	SGB_MEOH
235	NH3_HICO_CO	285	NH3_AQ3_H2	335	SGB_OTHERS
236	NH3_HICO_CO2	286	NH3_AQ3_CH4		
237	NH3_HICO_COS	287	NH3_AQ3_MEOH		
238	NH3_HICO_H3N	288	NH3_AQ3_OTHERS		
239	NH3_HICO_H2S	289	NH3_VENT_T_out		
240	NH3_HICO_H2	290	NH3_VENT_P_out		
241	NH3_HICO_CH4	291	NH3_VENT_H_out		
242	NH3_HICO_MEOH	292	NH3_VENT_N2		
243	NH3_HICO_OTHERS	293	NH3_VENT_O2		
244	NH3_LIQ1_T_out	294	NH3_VENT_H2O		
245	NH3_LIQ1_P_out	295	NH3_VENT_CO		
246	NH3_LIQ1_H_out	296	NH3_VENT_CO2		
247	NH3_LIQ1_N2	297	NH3_VENT_COS		
248	NH3_LIQ1_O2	298	NH3_VENT_H3N		
249	NH3_LIQ1_H2O	299	NH3_VENT_H2S		
250	NH3_LIQ1_CO	300	NH3_VENT_H2		

DEPARTMENT OF SPACE, EARTH AND ENVIRONMENT

CHALMERS UNIVERSITY OF TECHNOLOGY

GOTHENBURG, 2020