FACULTY OF TECHNOLOGY

Developing dynamic machine learning surrogate models of physics-based industrial process simulation models

Mikko Tahkola

Supervisors: Sorsa Aki, Paavola Marko

DEGREE PROGRAMME IN PROCESS ENGINEERING

Master's thesis

May 2019

ABSTRACT
FOR THESIS

University of Oulu Faculty of Technology

Degree Programme (Bachelor's Thesis, Master's Thesis)	Major Subject (Licentiate Thesis)
Degree Programme in Process Engineering	
Author	**Thesis Supervisors**
Tahkola, Mikko	Sorsa, A., D.Sc. (tech.) Paavola, M., D.Sc. (tech.)

Title of Thesis

Developing dynamic machine learning surrogate models of physics-based industrial process simulation models

Major Subject	Type of Thesis	Submission Date	Number of Pages
Automation Technology	Master's thesis	May 2019	94 p., 1 App.

Abstract

Dynamic physics-based models of industrial processes can be computationally heavy which prevents using them in some applications, e.g. in process operator training. Suitability of machine learning in creating surrogate models of a physics-based unit operation models was studied in this research. The main motivation for this was to find out if machine learning model can be accurate enough to replace the corresponding physics-based components in dynamic modelling and simulation software Apros® which is developed by VTT Technical Research Centre of Finland Ltd and Fortum. This study is part of COCOP project, which receive funding from EU, and INTENS project that is Business Finland funded.

The research work was divided into a literature study and an experimental part. In the literature study, the steps of modelling with data-driven methods were studied and artificial neural network architectures suitable for dynamic modelling were investigated. Based on that, four neural network architectures were chosen for the case studies. In the first case study, linear and nonlinear autoregressive models with exogenous inputs (ARX and NARX respectively) were used in modelling dynamic behaviour of a water tank process build in Apros®. In the second case study, also Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) were considered and compared with the previously mentioned ARX and NARX models. The workflow from selecting the input and output variables for the machine learning model and generating the datasets in Apros® to implement the machine learning models back to Apros® was defined. Keras is an open source neural network library running on Python that was utilised in the model generation framework which was developed as a part of this study. Keras library is a very popular library that allow fast experimenting. The framework make use of random hyperparameter search and each model is tested on a validation dataset in dynamic manner, i.e. in multi-step-ahead configuration, during the optimisation. The best models based in terms of average normalised root mean squared error (NRMSE) is selected for further testing.

The results of the case studies show that accurate multi-step-ahead models can be built using recurrent artificial neural networks. In the first case study, the linear ARX model achieved slightly better NRMSE value than the nonlinear one, but the accuracy of both models was on a very good level with the average NRMSE being lower than 0.1 %. The generalisation ability of the models was tested using multiple datasets and the models proved to generalise well. In the second case study, there were more difference between the models' accuracies. This was an expected result as the studied process contains nonlinearities and thus the linear ARX model performed worse in predicting some output variables than the nonlinear ones. On the other hand, ARX model performed better with some other output variables. However, also in the second case study the model NRMSE values were on good level, being 1.94–3.60 % on testing dataset.

Although the workflow to implement machine learning models in Apros® using its Python binding was defined, the actual implementation need more work. Experimenting with Keras neural network models in Apros® was noticed to slow down the simulation even though the model was fast when testing it outside of Apros®. The Python binding in Apros® do not seem to cause overhead to the calculation process which is why further investigating is needed. It is obvious that the machine learning model must be very accurate if it is to be implemented in Apros® because it needs to be able interact with the physics-based model. The actual accuracy requirement that Apros® sets should be also studied to know if and in which direction the framework made for this study needs to be developed.

Additional Information

TIIVISTELMÄ
OPINNÄYTETYÖSTÄ Oulun yliopisto Teknillinen tiedekunta

Koulutusohjelma (kandidaatintyö, diplomityö)	Pääaineopintojen ala (lisensiaatintyö)
Prosessitekniikan tutkinto-ohjelma	
Tekijä	**Työn ohjaajat yliopistolla**
Tahkola, Mikko	Sorsa, A., D.Sc. (tech.) Paavola, M., D.Sc. (tech.)

Työn nimi

Dynaamisten surrogaattimallien kehittäminen koneoppimismenetelmillä teollisuusprosessien fysiikkapohjaisista simulaatiomalleista

Opintosuunta	Työn laji	Aika	Sivumäärä
Automaatiotekniikka	Diplomityö	Toukokuu 2019	94 s., 1 liite

Tiivistelmä

Teollisuusprosessien toimintaa jäljittelevät dynaamiset fysiikkapohjaiset simulaatiomallit voivat laajuudesta tai yksityiskohtien määrästä johtuen olla laskennallisesti raskaita. Tämä voi rajoittaa simulaatiomallin käyttöä esimerkiksi prosessioperaattorien koulutuksessa ja hidastaa simulaattorin avulla tehtävää prosessien optimointia. Tässä tutkimuksessa selvitettiin koneoppimismenetelmillä luotujen mallien soveltuvuutta fysiikkapohjaisten yksikköoperaatiomallien surrogaattimallinnukseen. Fysiikkapohjaiset mallit on luotu teollisuusprosessien dynaamiseen mallinnukseen ja simulointiin kehitetyllä Apros®-ohjelmistolla, jota kehittää Teknologian tutkimuskeskus VTT Oy ja Fortum. Työ on osa COCOP-projektia, joka saa rahoitusta EU:lta, ja INTENS-projektia, jota rahoittaa Business Finland.

Työ on jaettu kirjallisuusselvitykseen ja kahteen kokeelliseen case-tutkimukseen. Kirjallisuusosiossa selvitettiin datapohjaisen mallinnuksen eri vaiheet ja tutkittiin dynaamiseen mallinnukseen soveltuvia neuroverkkorakenteita. Tämän perusteella valittiin neljä neuroverkkoarkkitehtuuria case-tutkimuksiin. Ensimmäisessä case-tutkimuksessa selvitettiin lineaarisen ja epälineaarisen autoregressive model with exogenous inputs (ARX ja NARX) -mallin soveltuvuutta pinnankorkeuden säädöllä varustetun vesisäiliömallin dynaamisen käyttäytymisen mallintamiseen. Toisessa case-tutkimuksessa tarkasteltiin edellä mainittujen mallityyppien lisäksi Long Short-Term Memory (LSTM) ja Gated Recurrent Unit (GRU) -verkkojen soveltuvuutta power-to-gas prosessin metanointireaktorin dynaamiseen mallinnukseen. Työssä selvitettiin surrogaattimallinnuksen vaiheet korvattavien yksikköoperaatiomallien ja siihen liittyvien muuttujien valinnasta datan generointiin ja koneoppimismallien implementointiin Aprosiin. Koneoppimismallien rakentamiseen tehtiin osana työtä Python-sovellus, joka hyödyntää Keras Python-kirjastoa neuroverkkomallien rakennuksessa. Keras on suosittu kirjasto, joka mahdollistaa nopean neuroverkkomallien kehitysprosessin. Työssä tehty sovellus hyödyntää neuroverkkomallien hyperparametrien optimoinnissa satunnaista hakua. Jokaisen optimoinnin aikana luodun mallin tarkkuutta dynaamisessa simuloinnissa mitataan erillistä aineistoa käyttäen. Jokaisen mallityypin paras malli valitaan NRMSE-arvon perusteella seuraaviin testeihin.

Case-tutkimuksen tuloksien perusteella neuroverkoilla voidaan saavuttaa korkea tarkkuus dynaamisessa simuloinnissa. Ensimmäisessä case-tutkimuksessa lineaarinen ARX-malli oli hieman epälineaarista tarkempi, mutta molempien mallityyppien tarkkuus oli hyvä (NRMSE alle 0.1 %). Mallien yleistyskykyä mitattiin simuloimalla usealla aineistolla, joiden perusteella yleistyskyky oli hyvällä tasolla. Toisessa case-tutkimuksessa vastemuuttujien tarkkuuden välillä oli eroja lineaarisen ja epälineaaristen mallityyppien välillä. Tämä oli odotettu tulos, sillä joidenkin mallinnettujen vastemuuttujien käyttäytyminen on epälineaarista ja näin ollen lineaarinen ARX-malli suoriutui niiden mallintamisesta epälineaarisia malleja huonommin. Toisaalta lineaarinen ARX-malli oli tarkempi joidenkin vastemuuttujien mallinnuksessa. Kaiken kaikkiaan mallinnus onnistui hyvin myös toisessa case-tutkimuksessa, koska käytetyillä mallityypeillä saavutettiin 1.94–3.60 % NRMSE-arvo testidatalla simuloitaessa.

Koneoppimismallit saatiin sisällytettyä Apros-malliin käyttäen Python-ominaisuutta, mutta prosessi vaatii lisäselvitystä, jotta mallit saadaan toimimaan yhdessä. Testien perusteella Keras-neuroverkkojen käyttäminen näytti hidastavan simulaatiota, vaikka neuroverkkomalli oli nopea Aprosin ulkopuolella. Aprosin Python-ominaisuus ei myöskään näytä itsessään aiheuttavan hitautta, jonka takia asiaa tulisi selvittää mallien implementoinnin mahdollistamiseksi. Koneoppimismallin tulee olla hyvin tarkka toimiakseen vuorovaikutuksessa fysiikkapohjaisen mallin kanssa. Jatkotutkimuksen ja Python-sovelluksen kehittämisen kannalta on tärkeää selvittää mikä on Aprosin koneoppimismalleille asettama tarkkuusvaatimus.

Muita tietoja

PREFACE

This thesis is made for VTT Technical Research Centre of Finland Ltd and as a part of master's degree programme in process engineering in University of Oulu. The objective was to study feasibility of applying machine learning methods in creating computationally lighter surrogate models of dynamic physics-based industrial process simulation models. The main motivation for this study is to find out if the simulation speed of a simulator can be increased by replacing computationally heavy parts of it with models created using machine learning methods. The work was executed in EU funded COCOP project and Business Finland funded INTENS project and thus, EU and Business Finland are acknowledged.

I want to thank Jouni Savolainen and Juha Kortelainen from VTT for their guidance and comments throughout the thesis project. In addition, big thanks goes to each VTTer who have given me practical advices during the study. I also want to thank postdoctoral researchers Aki Sorsa and Marko Paavola from University of Oulu for supervising the thesis work and especially Aki Sorsa for all the comments.

Oulu, 22.5.2019 Mikko Tahkola

CONTENTS

1 INTRODUCTION

Mathematical modelling and simulation are important tools in different phases of process and automation engineering projects. These tools can be used for example in creating new information that can be valuable in the process design tasks and in optimising existing processes. Dynamic modelling and simulation also fall into this class of tools and can be utilised in e.g. building a virtual version of industrial processes. These virtual plants are used in e.g. process operator training as it is more risk free than training on a real plant, and all kinds of new operating and control methods can be developed without fear of accidents. (Lappalainen et al. 2012).

Running physics-based dynamic process simulation can be computationally expensive when the simulation models are large and include high amount of details. Simulation speed of this kind of models might be slower than real-time which can be a problem when the model is needed for example in operator training, plant design or optimisation tasks. Especially when using a simulator for operator training it is a requirement for the simulator to run in real-time. The main objective of this research is to study feasibility of building computationally light dynamic machine learning surrogate models of physics-based unit operation models that are built in dynamic process modelling and simulation software Apros. This is to see if the machine learning model could be implemented in Apros to replace computationally heavy components of the model and this way increase the simulation speed to make the model usable in more applications. In this work, data-driven models created using machine learning methods are referred to as machine learning models. Different kinds of models can be derived using machine learning methods but in this study, the scope is in using artificial neural networks.

A lot of literature is available for data-driven dynamic simulation, often referred to as multi-step-ahead predicting or forecasting, e.g. (Taieb and Hyndman 2012), (Sun et al. 2010) and (Papacharalampous et al. 2019). However, combining physics-based and machine learning models in a dynamic modelling and simulation environment is a less studied topic. Hybrid modelling in process engineering is discussed e.g. in (Stosch et al., 2014) in which also references to literature on utilising artificial neural networks together with physics-based models are given.

Apros is a so called first principle modelling and simulation tool in which physical and chemical laws and principles are taken into account. Including machine learning models as black-box components is in the scope of interest not only to increase the simulation speed but also to enable possibility of hiding confidential information included in the model from the simulation model user without affecting other parts of the simulation model. This research is carried out as a combination of literature and case studies. The literature part gives an introduction in topics such as modelling of dynamic systems, physics-based modelling and finally dynamic data-driven modelling and the steps included in it. Artificial neural networks, being the approach for data-driven dynamic modelling in this study, are presented in more detail although the steps of the modelling can be applied in most other data-driven model types as well.

In the experimental part, the workflow from selecting one ore multiple unit operation models to be modelled using machine learning methods to finally implementing the data-driven surrogate model back to Apros is defined. The main focus in this research is in the data-driven modelling using artificial neural networks and for that, a framework to build machine learning models was implemented. The framework includes steps from pre-processing to model selection. Data for the models is generated in Apros and the data-driven models in this study are created using Keras (Chollet 2015). The studied model types are linear and nonlinear versions of an autoregressive neural network model with exogenous inputs (ARX and NARX respectively), Gated Recurrent Unit (GRU) neural network and Long Short-Term Memory (LSTM) neural network. Although the way to implement Python-based machine learning models in Apros is defined in this study, that part of the process requires more work in order to get the machine learning models work properly in Apros together with the physics-based model.

This research has been executed in COCOP and INTENS projects. The COCOP project receive funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 723661. In COCOP project, the objective is to define, design and implement a concept that integrates existing industrial control systems with efficient data management and optimisation methods. The objective is also to provide methods to monitor and control large industrial processes. (Anon 2019f) Simulation models are utilised which speed requirement is high in the optimisation tasks. In this study, machine learning methods to improve the speed of the simulation models are investigated. INTENS project is Business Finland funded project that focus on

advancing, promoting and digitalising Finnish marine industries. Special focus in INTENS is on improving energy efficiency and decreasing emissions of ship energy systems. (Anon 2019g)

2 MODELLING OF DYNAMIC SYSTEMS

Dynamic process models can be used e.g. in process design, process control design and testing, process optimisation and fault detection purposes (Ikonen and Najim 2002, p. 3–4). A system can be described as a group of components that interact with each other. Current state of a system is defined by state variables, which contain information about the system. Factors that externally affect the behaviour of a system are called inputs. The input and state variables can be used to define the next state and the output of the system. (Stanislaw 2003, p. 1) For example, a sequence of three water tanks is defined as a system by choosing the boundaries so that the system begins from the inlet pipe of the first tank and ends at the outlet pipe of the third tank, as shown in Figure 1. Any of the three tanks can also be considered as a subsystem of the original system or as a system on its own. Considering the sequence of tanks, we could define the system boundaries so that the mass flow in the inlet pipe to the first tank acts as an input and the outlet flow from the third tank acts as an output. In this example, water levels in the tanks could be selected as state variables.

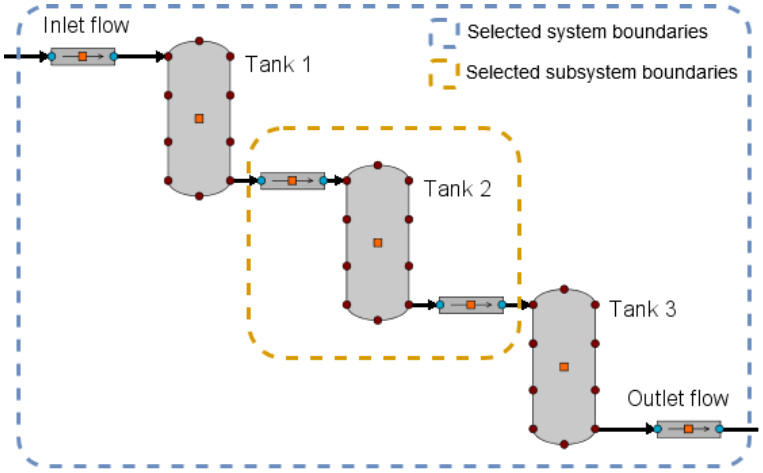


Figure 1. System of three sequential water tanks (blue boundary) and tank 2 defined as a subsystem (orange boundary).

Systems can be classified in many ways, for example to linear and nonlinear. A system, H, is linear if it obeys superposition principle, which is mathematically described such that (1) is true for any constants c_1 and c_2, and inputs $x_1 = x_1(t)$ and $x_2 = x_2(t)$. By the same principle, there can be any number of vectors on left hand side and longer combination on right hand side in (1) when the system is linear. A system is nonlinear if it does not

comply with one or both properties of superposition principle. The superposition principle is given by: (Bendat 1998, p. 1-2)

$$H[c_1x_1 + c_2x_2] = c_1H[x_1] + c_2H[x_2]. \quad (1)$$

This means that the relations between input and output of a linear system are additive (2) and homogenous (3). (Bendat 1998, p. 1–2)

$$H[x_1 + x_2] = H[x_1] + H[x_2] \quad (2)$$

$$H[cx] = cH[x] \quad (3)$$

A system can also classified by whether it is static or dynamic. Behaviour of a static system is not dependent on the previous external stimuli but only on the current, i.e. if there is change in the input, the output will change immediately. In dynamic systems, values of output variables may change even when there are no changes in the external stimuli, i.e. the output depends on the past in addition to current stimuli. (Ljung and Glad 1994, p. 19) As stated in (Haykin 2009, p. 797), the state of a dynamic system includes all the necessary information about the earlier behaviour of the system that can be used together with external inputs to predict the next state and output. Whether the system is static, dynamic or classified in some other way, we can use mathematical equations to define relationships between the components of a system and we get a mathematical model of the system (Ljung 1987, p. 5).

Beside division between nonlinear and linear, and static and dynamic, process models can also be divided into physics-based (white-box), data-driven (black-box) and hybrid (grey-box) models. In surrogate modelling, there is already an existing model of a system, but it may be originally designed for other purposes and does not meet requirements for some other use and therefore a surrogate model is needed. For example in this research, the main motivation to study suitability of data-driven methods in modelling dynamic industrial processes is to find out if physics-based unit operation model of a process model in Apros can be replaced with a data-driven surrogate. Physics-based modelling is discussed in Section 2.1 and Apros is presented in Section 2.2.

2.1 Physics-based modelling

Physics-based, which are also called first principle models, are built of mathematical equations, which have proved to explain the physical laws and principles describing the behaviour of the process (Ikonen and Najim 2002, p. 4). Modelling begins with structuring the problem that is dividing it into smaller problems, or the system to subsystems. The purpose of the model is taken into consideration at the very beginning. Inputs, outputs, variables and constants of the system or subsystem are defined and time-dependency of those is looked into. In addition, the relations between variables and constants are examined to define if are they static and dynamic. (Ljung and Glad 1994, p. 83–84) Next, the known physical laws are applied to the variables and constants in the subsystems to form relations between them. Physical relations can be divided to conservation laws like conservation of mass and energy and constitutive relations that for example define how the ratio of input and output flows in a tank system is related to the water level in the tank. (Ljung and Glad, 1994, p. 92)

Usually some assumptions are also required in order to simplify the model enough for the purpose it is going to be used for. Simplification of a model can be made in multiple ways. Firstly, variables that has only small effect on the system can be neglected or a compressible fluid could be assumed incompressible to simplify the model. In addition, the complexity can be reduced by choosing time constants by the most interesting phenomena in the system and simplifying the subsystems that haves significantly slower or faster dynamics. This can be made by replacing significantly smaller time constants by using static relationships and bigger time constants by using constants. When there are time constants considerably different in values with each other, stiffness of differential equations representing the system increases. Stiffness makes numerical solving of differential equations less efficient. (Ljung and Glad 1994, p. 98–101)

Now that the relations are defined, one can choose state variables for the system. In the earlier tank sequence example, water levels in the tanks were proposed as state variables. To be able to calculate the next states and outputs of the system, time-derivatives of the state variables must be formed as a function of previous state and input. To simplify the model, state variables could be aggregated that practically means reducing the amount of similar state variables by combining them. (Ljung and Glad 1994, p. 100–101 & 105)

2.2 Apros® – Dynamic process modelling and simulation software

Apros is a physics-based advanced dynamic process modelling and simulation software developed by VTT Technical Research Centre of Finland and Fortum since 1986. There are multiple Apros products – Apros Thermal, Apros Nuclear, Apros Nuclear 3D + Containment and Apros Pulp & Paper, each targeted for modelling different processes. Each of the versions include all the thermal hydraulic solvers, water-steam material properties, basic process components, automation and electrical components and data logging and visualization features. The nuclear version includes also nuclear process components and 1D reactor model whereas in the Nuclear 3D + Containment version there are also 3D reactor and containment models. The Pulp & Paper version adds process components related to pulp and paper industry to the Thermal version as the name suggests. (Anon 2019a)

Apros can be used for modelling and simulation of e.g. nuclear power plants, thermal power plants, and pulp and paper processes including their automation and electrical systems. It is a tool that can be utilised e.g. in process and control design, optimisation and automation system testing tasks. In addition, it can be used in analysing the processes, troubleshooting tasks, training purposes and to execute safety analysis. (Anon 2018a) The model building in Apros is based on using ready-made basic component models such as pipes, pumps, tanks and heat exchangers that are included in a symbol library. These component models in Apros are called modules and this term is also used in this thesis from now on. The modules are implemented in models by dragging-and-dropping them from a symbol library to a diagram in the graphical user interface (see Figure 2).

Automation modules include e.g. controllers, actuators and measurement modules. Alternating and direct current systems can be modelled with included electrical modules like generator, transformer or battery. Custom modules can be built by using User Component and External Model modules. By utilising External Model module, the user can also include C, C++ or FORTRAN code in form of a DLL file to add custom calculations. Once Apros solver encounters an external model, it will call a predefined function in which the functionality of the model is stored. (Anon 2018b) User Components are presented in Subsection 2.2.1.

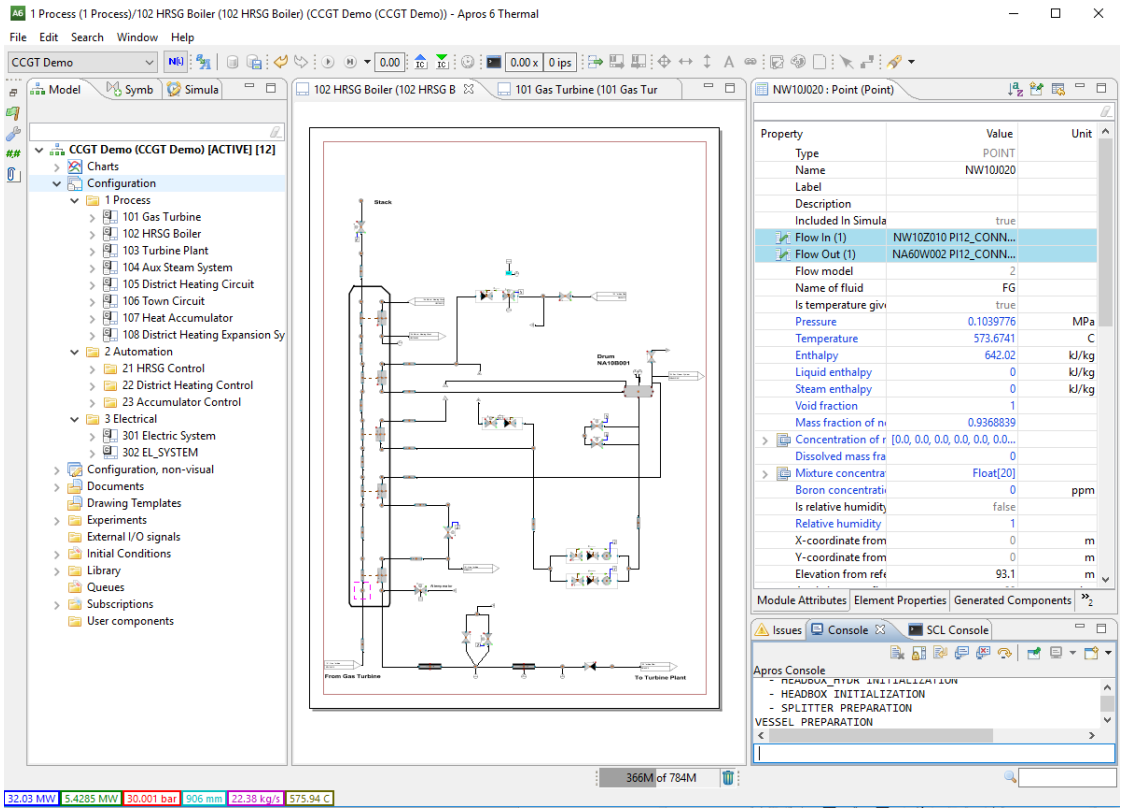


Figure 2. Graphical user interface of Apros.

Mechanistic models in Apros combine structure of the modelled process, first principles of physics and chemistry, empirical correlations such as heat transfer correlations, and properties of a set of materials. During simulation, in addition to partial and ordinary differential equations, nonlinear and linear algebraic equations are solved at each time step. (Anon 2018a) Apros includes multiple flow models that are computed differently. The six-equation flow model is the most extensive flow model in Apros and includes one-dimensional mass, momentum and energy equations for liquid and gas phases which results in six partial differential equations. These equations are first discretised and nonlinear components are linearised. The discretisation is made with respect to space and time with staggered grid method in which the values of state variables like pressure and enthalpy are calculated in the center of the grid's cells and e.g. velocities of the fluid flows two phases are calculated in the edge of the cell. In numerical solution, implicit method is used iteratively until convergence and in each iteration, pressure, void fraction and enthalpies of the two phases are calculated sequentially. Liquid and gas properties like density and temperatures are calculated based on pressure and enthalpy. (Hänninen and Ylijoki 2005) The simulation time step can change dynamically in Apros. For example,

if fast transients occur, a shorter time step may be needed to increase the accuracy until the process gets back to more stable operation, when the time step can be extended. (Anon 2019b).

2.2.1 Apros User Components

As mentioned, in addition to using modules in the symbol library presented earlier, the user can create re-usable and customizable modules called User Components that can be built of symbol library modules and other user components. SCL scripts (Simantics Constraint Language) can also be attached to User Components to include custom calculations. User Components can be built similarly as model diagrams are built, i.e. by dragging and dropping modules from the symbol library to the User Components diagram. Data transfer between User Component and other Apros modules can be done in multiple ways. One way is to create terminal connection points for the User Component symbol in which input and output signals can be connected. In this study, however, the User Component module is utilised in a way that terminal connection points are not needed. All the functionality of the User Component, i.e. the data-driven model, is included in SCL scripts that are attached to the User Component. Data from physics-based Apros modules can be read directly to SCL variables and also after performing operations using SCL, the values of SCL variables can be written directly to properties of Apros modules.

2.2.2 Python binding in Apros

Python binding was used in Apros in this work to enable the use of data-driven models written in Python. The binding allows execution of Python statements within SCL. In practice this means that user can assign values from process components, e.g. mass flow in a pipe to a Python variable using SCL. The Python variable can then be used during execution of Python statements and the output can be assigned back to SCL variable to be used in Apros. The use of Python binding in implementing data-driven models in Apros is explained in Chapter 5 in detail.

2.2.3 Logging input and output data

Apros includes options to write and read data between Apros models and text files during the simulation. In this research, values of the input and output variables are written to text

files during dataset generation simulation run. The data logging is set up before generating datasets for data-driven modelling by defining the modules and the properties of the modules, i.e. the input and output variables, which need to be logged. Also, the sampling time of data logging can be defined. In this study, the sampling time was 0.2 s in both case studies which are presented in Sections 5.2 and 5.3. Data transfer can be controlled with SCL commands which are included in the SCL script that is used to run the experiments.

3 DYNAMIC MODELLING WITH ARTIFICIAL NEURAL NETWORKS

Data-driven modelling is a method of system identification that is based on input-output data extracted from the system. Creating a data-driven model includes several steps, which are the same whether the modelled system is static or dynamic. The first step is to gather the data that describe the behaviour of the system. Next, a model structure that is suitable for the case is chosen. The model gets its shape when the model is fitted with data which practically means that the parameters of the selected model structure are estimated so that the input-output behaviour of the model is sufficient. In other words, one or more functions are approximated during the identification process so that they represent the input-output relationships with sufficient accuracy. Final step is to measure the performance of the model which is called model validation. Performance requirements for models may vary but often we want the model to be as accurate as possible and generalise well at the same time. However, the purpose in which the model is used may also set additional requirements such as high computational efficiency. (Ikonen and Najim 2002, p. 9–11)

There are numerous different methods available to be applied in every step of the identification process. Choosing a suitable model type is essential to capture the characteristics and properties of modelled process. When dealing with dynamic systems, output of the system often does not only depend on the current input but also on the past inputs, outputs and states of the system. (Ikonen and Najim 2002, p. 113). In this chapter, the process of modelling with artificial neural networks is explained beginning from the data generation and pre-processing to model selection. Although choosing the model type is a part of the modelling process, the different recurrent artificial neural network architectures suitable for dynamic modelling are presented in Chapter 4 together with introduction to autoregressive models and artificial neural network models in general.

3.1 Designing the experiments for data generation

To be able to create machine learning models based on data generated in Apros, an experiment design needs to be made or selected in order to first generate the data. An experiment design is a plan that is used to create data of a system in a way that the

relations between inputs and outputs of the system become visible in the data. During the dataset generation, changes are caused in the values of excitation variables, i.e. the input variables of the system, which cause some kind of effect in the observation variables, i.e. the outputs of the system. There are multiple things to take into account when choosing the experiment design. The most important thing is to consider the use for what the data-driven model is to be constructed. (Sun and Sun 2015) When making a data-driven surrogate model to be used in dynamic simulation environment, the coverage and reliability requirements for the model are high.

3.1.1 Experiment designs

Full factorial experiment design includes two or more levels for each input factors. In a two-level full factorial design, the levels are often called *low* and *high*, or *-1* and *1* level. The number of experiments in a full factorial design depends on the number of input factors and the number of levels. Each combination of the input factors on different levels are included in the design. Full factorial design grows very large quickly as the number of input factors and their levels increases and for example in (NIST/SEMATECH 2012) it is recommended not to use it if the number of input factors is more than 4. However, this is not necessarily a problem when the experiments are made in simulation environment which is the case in this study, although it increases the time to develop the models. To decrease the amount of needed experiments, fractional factorial experiment designs can be used. Fractional factorial designs include only a part of the experiments of full factorial design. (NIST/SEMATECH 2012)

Central Composite Design (CCD) is another experiment design that has multiple different configurations (see Figure 3 for illustration of case with two input factors). Central Composite Circumscribed (CCC) design includes five levels which are called the edge points, the star points and the center point. Edge points are the same as the low and high level in a factorial design. Distance of the star points is related to the number of input factors that are included in the design. In Central Composite Inscribed (CCI) design there are also five levels, and the structure is the same as in CCC, but the star points are located at the same locations where the edge points are in CCC, i.e. they are closer to the center points and the operation range is narrower than in CCC. Central Composite Face centered (CCF) design on the other hand have only three levels. (NIST/SEMATECH 2012)

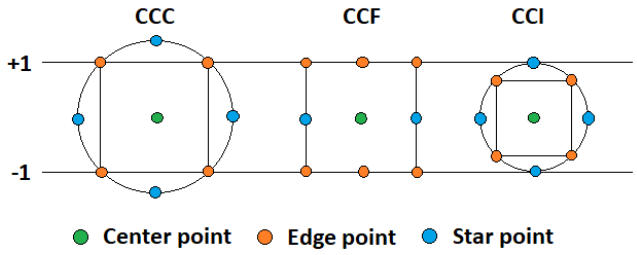


Figure 3. The three types of Central Composite Design of experiments. Adapted from (NIST/SEMATECH 2012).

Box-Behnken (BB) experiment design, illustrated in Figure 4 for three input factors, is a quadratic design that require less experiments than full factorial design. It also requires less experiments than CCI or CCC when the number of input factors is less than five. In BB design, there are three levels which are at the midpoints of edges and in the center. Therefore it does not include the corner points which means that input factor extremes are excluded from the experiments. (NIST/SEMATECH 2012)

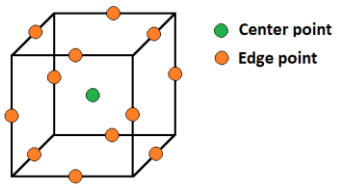


Figure 4. Illustration of Box-Behnken experiment design for three input factors. Adapted from (NIST/SEMATECH 2012).

Modelled processes in Apros are dynamic, include multiple variables and often include nonlinearities with different kind of interactions between the variables. Thus a two-level experiment design does not necessarily provide enough information to reveal the possible nonlinearities which requires three-level or higher level experiment design. The principle for this is demonstrated in Figure 5 A and C. In 5A, there are two levels used in experiment design and as a result, the relation between variables appears to be linear. However, if the third point is added to the middle point in experiment design, the results may reveal that the relation is actually nonlinear.

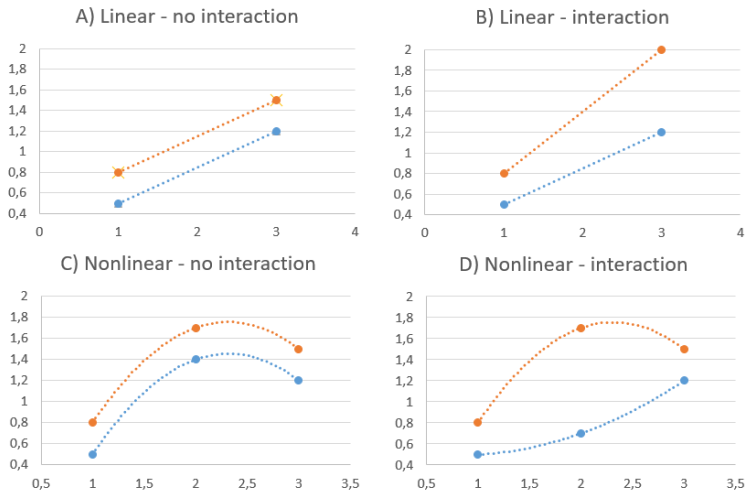


Figure 5. Demonstration of differences between different relationships of two variables. Adapted from (Leiviskä 2013).

If behaviour of the modelled output variables is cubic (see Figure 6), the experiment design should contain at least four levels. It can be concluded from this section, that the selection of a suitable experiment design is of high importance when a data-driven model of a system is to be built. In the first case study of this research, a tailor-made experiment design was used as there was only one input factor. In the second case study, CCI design that include five levels, was used in the experiments as there was five input factors and experiments with full factorial design would have led to impractically high amount of data for developing models. CCI was chosen over Box-Behnken design as it includes more levels, i.e. more complex relationships can be revealed if there are some to be found. Wide range of other experiment designs can be found from literature apart from the ones presented in this section (see NIST/SEMATECH 2012).

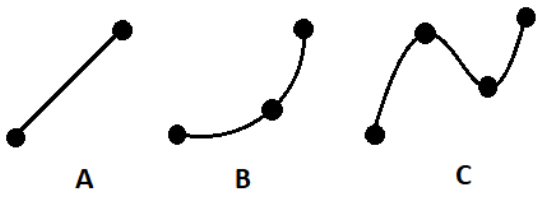


Figure 6. Linear (A), quadratic (B) and cubic (C) functions demonstrating how many levels are needed in experiment design for different kind of functions that are modelled. Adapted from (NIST/SEMATECH 2012).

3.1.2 Importance of sampling rate

Sampling rate, or sampling time, defines how often the data is stored during the experiments. Choosing a suitable sampling rate for a specific case is an important thing to consider when a data-driven model is to be built. Sampling theorem of Nyquist-Shannon, that was originally introduced in (Shannon 1949), states that to be able to reconstruct the original continuous-time signal from data sampled in discrete time steps, there is a limit for the discrete-time sampling frequency. According to the theorem, the sampling frequency must equal at least the signal's highest frequency multiplied by two, i.e. $f_s \geq 2f_h$, where f_s is the required sampling frequency and f_h is the highest frequency of the original signal. If the sampling frequency is lower than the Nyquist-Shannon theorem states, the discrete-time sampled version of the original signal will be distorted due to aliasing or frequency-folding. (Van den Hof 2012)

In this study, the data is in time series form and is sampled with a constant frequency during the data generation run in Apros. Sampling time and simulation time step in Apros are not required to be equal even though this is the case in this study, i.e. the simulation time step and sampling time is 0.2 s in the case studies. This mean that all information calculated by Apros is stored during the experiments in the datasets and no data loss occurs. However, in the second case study the amount of data was reduced by down sampling the data from 0.2 s sampling time to 5 s in order to keep the neural network training times in reasonable limits. The training time of a neural network depends on the amount of data used and thus choosing unnecessarily low sampling time would make the training process slower even though longer sampling time would have been enough to reveal the dynamics of the process. By decreasing the sampling time by a factor of two, e.g. from 2 s to 1 s, the amount of data increase by the same factor of two.

3.2 Pre-processing of data

Pre-processing of data is an important step to perform before data-driven modelling. In cases where the system consists of multiple input and/or output variables, there may be variables whose units are different and have differences in their orders of magnitude. Without pre-processing, modelling becomes difficult because values of variables with low order of magnitude and range seem to vary less than the variables with higher order of magnitude. Consequently, the model emphasises the latter more. In normalization, the

scale of each variable is modified so that each variable has the same scale. (Dreyfus 2005) In data-driven modelling it is often called min-max feature scaling where final scale is often -1–1 or 0–1. Another data pre-processing procedure used often in data-driven modelling is standardization. In standardization, the data is processed so that each of the variables have mean of zero and standard deviation of one (Graves 2012). According to (Graves 2012), standardization of input data can affect the performance of a machine learning model significantly. However, when modelling dynamic industrial processes, the machine learning models need to include an autoregressive component, i.e. previous output values are used in the input, also the output values need to be pre-processed accordingly.

Other time series data related pre-processing methods are detrending and deseasonalising which remove trend or seasonal effects from data to make it easier to model. Seasonal effects occur periodically. For example, outdoor temperature has a seasonal component on a yearly scale as it is lower during winter and higher during summer on average. Trend in time series values means that the values are increasing or decreasing over time. When using a model trained on detrended and/or deseasonalised data, the trend and seasonal component are included back to the prediction. (NIST/SEMATECH 2012)

Data from Apros can be considered noiseless and in that point of view, training a machine learning model to fit the data becomes easier than it would be with noisy data. Furthermore, it also reduces pre-processing needs as there are e.g. no outliers or missing values in the data that would need to be taken care of before modelling. It is of great importance to notice that the validation and testing datasets need to be pre-processed in the same way as the training dataset. For example, if input variable *pressure* is scaled into range -1–1 by using its minimum and maximum values in the training set, the same minimum and maximum values must be used when scaling the values of input variable *pressure* in the validation and testing sets as well.

3.3 Input variable selection

When building a data-driven model, the dataset may consist of tens, hundreds or even more candidate input variables that are measured or created by using feature engineering methods. The purpose of input variable selection, or input feature selection, is to find the most relevant variables of these candidates to be included into the model and also to

exclude irrelevant and redundant variables from the input data. Input variable selection has an important role in data-driven modelling regardless of the modelling technique. Irrelevant variables are variables that do not provide any predicting power to the model which means that they only cause noise and increase the amount of model parameters without making model performance better. Redundant variables may be relevant but do not increase the predicting power of the model because some other variable provides the same information. The selected input variables should provide information that is needed to explain the behaviour of the outputs with required accuracy and provide good generalisation ability at the same time. (Galelli et al. 2014). In (May et al. 2011) it is also stated that when dealing with artificial neural networks, the input variable selection is important because redundant and irrelevant input variables make the training process harder as they increase the amount of local optimum solutions in the error function. This is problematic especially with training algorithms based on gradient descent as it becomes more probable that they converge to a local optimum and thus the generalisation ability of the model would be weak. (May et al. 2011)

In feature engineering, new features are generated from the existing features and these new features are also taken into consideration in input variable selection. When dealing with time series data, this may include generating lagged versions of the original inputs. Therefore, the input variable selection problem expands in time series cases as the optimal number of lags, i.e. the number of values from previous time steps, must also be determined. Increasing number of lagged variables increases the number inputs which practically leads to a more complex model. (Bowden et al. 2005)

Input variable selection methods can be divided in two classes – model-based methods and filter methods. Filter methods do not require creation of models whilst the model-based input variable selection methods include a model to measure the performance of each selected input variable combination. Model-based methods include two subclasses which are wrapper methods and embedded methods. (Galelli et al. 2014) & (Guyon et al. 2006, p. 6–7) Different search strategies can be utilised to find the best input variables when there are too many possible combinations to try separately (Guyon et al. 2006, p. 6–7). When modelling complex nonlinear processes with artificial neural networks, it is important to choose an input variable selection method that is able to take into account the nonlinearities and redundancy of the input variables, i.e. if there are two or more candidate variables that provide the same information, only one is selected. (May et al.

2011) This kind of methods are listed in Table 1 and are discussed in the following Subsections 3.3.1–3.3.2.

Table 1. Input variable selection methods suitable for nonlinear systems taking redundancy into account. (May et al. 2011).

Input variable selection method	Type
Forward selection	Wrapper
Backward selection	Wrapper
GA-ANN (Genetic Algorithm - ANN)	Wrapper
MIF (Mutual Information Forward)	Filter
PMI (Partial Mutual Information)	Filter
RFE (Recursive Feature Elimination)	Embedded
EANN (Evolutionary ANN)	Embedded

For linear systems, the selection of the number of lagged values to be included in the model can be estimated for example by calculating autocorrelation for the output variables and cross-correlation of input and output variables. (Bowden et al. 2005) According to (Yu et al. 2000), selection of the input variables and determining the order and time-delay of the modelled system are often carried out by building models with different combinations of these and selecting the one with the lowest error. They argue that the approach is not easy to use with complex industrial processes that include multiple variables and present an alternative in which the orders and delays are found out by building simpler linearized models of a nonlinear process in multiple operating points. (Yu et al. 2000)

Reinforcement learning has also been used in estimating optimal input variables and time delay between the input and output variables in (Liu et al. 2005). Reinforcement learning is out of the scope of this research although it has been applied to time series modelling problems. For example in (Venkatraman 2017), the author of the thesis introduces a new training algorithm that can be used in time series modelling and shows that multi-step-ahead predicting performance with recurrent neural networks can be improved. In the introduced Data as Demonstrator algorithm, the training data is used to generate synthetic training examples which can be used in correcting the multi-step-ahead predictions. (Venkatraman 2017)

3.3.1 Wrapper methods

Wrapper methods apply a model to measure the performance that the selected input variables provide and therefore the efficiency of this method is related to the model's capability to describe the data it is trained with. The requirement of building a separate model for each selected input variable combination also has an impact on the time that is needed to find the best combination. Using nonlinear models in the process enables accounting for nonlinear relationships between the input and output variables.

Stepwise variable selection methods are simple techniques in which either none (forward selection) or all (backward elimination) input variables candidates are used initially in the model. In forward selection, a separate model is first built for each of the input variable candidates and the candidate that explains the data better than others, is selected to the next round. During the following rounds, variables that are not selected yet are tested one by one in model building together with the already selected input variable or variables. In backward elimination, the process is reversed as initially all the input variable candidates are used in the model. At each round one variable at a time is eliminated to find out which is the least promising considering the model performance. This least promising candidate is discarded, and the process continues to the next round. (May et al. 2011)

Genetic algorithms combined with artificial neural network have also been used in input variable selection by (Bowden et al. 2005). In their approach, a hybrid genetic algorithm and generalised regression neural network (GRNN) is used to find the best input variables and as a pre-processing step, a self-organizing map (SOM) is used to reduce the number of candidate input variables and to find independent inputs. Self-organising maps can be used for finding redundant variables in both linear and nonlinear cases. (Bowden et al. 2005)

3.3.2 Embedded methods

In embedded methods, the input variable selection is included in the training process of artificial neural networks. The difference between the embedded and wrapper methods is that the wrapper methods take into account only the model performance with selected input variables as a group whereas an embedded algorithm takes into account how the input variables affect the model performance separately. Recursive feature elimination (RFE) is one embedded algorithm which is based on backward elimination. In RFE, first

the models are built using all the input variables and which are then dropped one by one as in backward elimination procedure. Evolutionary artificial neural network (EANN) is also classified as embedded method. Gradient descent is not used in the training of EANN, but the weights of EANN are optimised using some evolutionary algorithm (EA). In (May et al. 2011) this approach is suggested to be more robust in converging to near-global optimum than gradient descent. (May et al. 2011)

A hybrid algorithm called GASA-RBF was proposed in (Alexandridis et al. 2005) which combines genetic algorithm with simulated annealing and radial basis function (RBF) neural network, being suitable for nonlinear systems. In their approach, genetic algorithm is first used to find an input variable combination that minimises prediction error of RBF network and then Generalised Simulated Annealing algorithm is used to find out if the amount of input variables can be reduced. (Alexandridis et al. 2005)

3.3.3 Filter methods

Filter methods in input variable selection differ from embedded and wrapper methods by being a model-free technique. Filter methods can be divided in linear correlation-based methods and methods based on information theoretic measures. Linear correlation is a commonly used input variable selection method. For example in equation $y = 2x$, variables x and y are correlated linearly. (May et al. 2011) & (Anon 2019h) Mutual information (MI) and partial mutual information (PMI) are filter methods that are suitable for nonlinear cases. MI and PMI methods are based on probability distributions, i.e. they do not make assumptions of the structure of the relationship of the variables (i.e. linearity/nonlinearity). With the PMI method, the significance of input candidates can be found out. (May et al. 2011)

3.4 Neural network training

Training of a neural network means adjusting the parameters, i.e. the weights, of the neural network in a way that the neural network model fits well on the data it has been developed with (i.e. training data). Neural network must also have the ability to generalise well which means that it should perform well also on data that is different from the data used in the training phase. Neural network training is performed usually inside a hyperparameter optimisation loop, where multiple of models are trained with different

hyperparameters to find the optimal ones. Hyperparameter optimisation is presented in Section 3.5.

Machine learning, including the training of neural networks, can be divided in multiple classes according to the problem set up and how the model is built. When there is data for both input and output variables that match each other, the learning process is called supervised learning. Thus, supervised learning is based on comparing predicted output values with ground truth output values and the model is adjusted in a way that the error between these two is minimized. (Graves 2012) In unsupervised learning, there is no information of the outputs and the learning is based on observing features from data. Clustering is one example of an unsupervised learning problem, where a set of data points are divided into groups. This is made by measuring pairwise dissimilarities of two data points in the same cluster and in other clusters. Dissimilarity of two data points in the same cluster is smaller than that of data points belonging to different clusters. (Hastie et al. 2009) When the learning is based on minimising or maximizing a cost function instead of comparing predicted values with ground truth values as in supervised learning, we are talking about reinforcement learning. Reinforcement learning is applied e.g. when it is too difficult to obtain input and output values of the problem for modelling. (Graves 2012) In this study, input and output pairs are known and thus supervised learning methods are applied.

Training of complex artificial neural networks require considerable amount of calculation resources. During last decades, the calculation capabilities of computers have increased significantly, allowing artificial neural networks to be used more and more as a nonlinear modelling tool. (Dreyfus 2005 p. 1) Recurrent neural networks are often trained using a technique called teacher forcing. In this technique, the true outputs are used in the input of the network whilst when using the model, the actual output of the model is used. (Haykin 2009, p. 817). The teacher forcing technique is demonstrated in Figure 11 in Subsection 4.2.1. When using the trained model, the calculated output signals are fed back to the inputs instead of true values. The teacher forcing technique can be applied in other model types as well if they have autoregressive inputs. Different autoregressive model structures are presented in Chapter 4.

The training process of neural networks can be monitored by plotting the training and validation errors on each epoch as illustrated in Figure 7. In Figure 7, the validation loss

begins to eventually increase after certain number of training epochs, i.e. training iterations, have passed. This implicates overfitting which means that the network has learned the training data too well and is not able to generalise, i.e. the model does not work with any other data than the original training data. Therefore, it can be useful in some cases to use the so called early stopping technique to find optimal generalisation ability of the network. When early stopping is applied in the training process, the validation loss is measured after each training epoch and based on the method, the training can be stopped by different criteria, e.g. if the validation loss starts to grow implicating overfitting or if the validation loss has not decreased during the last N epochs. (Orr and Müller 1998) However in this study, the early stopping was not found useful because the relationship between one-step-ahead (or input-output mapping) and multi-step-ahead validation loss seems to be uncorrelated. On the other hand, there are other ways of addressing overfitting in addition to early stopping, called regularisation methods. Some regularisation methods are presented in Subsection 3.4.2.

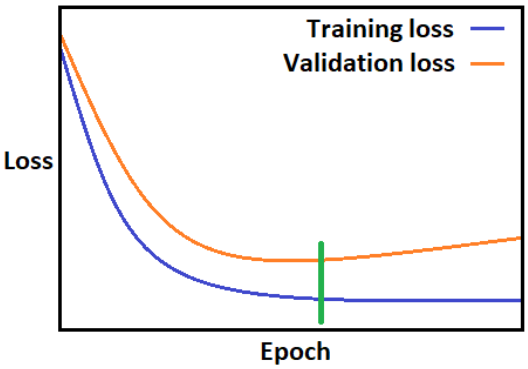


Figure 7. Illustration of how training and validation loss may behave during the training of neural network. The model in this example would be over fitted if the training is continued after the epoch marked by the vertical green line, i.e. if the validation loss starts increasing. Adapted from (Orr and Müller 1998)

3.4.1 Gradient descent based weight optimisation

Algorithms based on gradient descent are used widely in optimizing weights of a neural network during the training. The efficiency of gradient descent optimisation stems from the need to only calculate first-order partial derivatives (Kingma and Ba 2014). In gradient descent optimisation algorithm, finding the global minimum of an objective function is carried out by calculating the gradient of the objective function, i.e. the

gradient of the loss function in neural network training. The parameters are then updated in the direction of negative gradient, i.e. in the direction in which the loss decrease the most. How much the values of parameters are updated depends on parameter called step size that is called the learning rate in neural network training. Backpropagation is a popular gradient descent based method and an efficient way of calculating gradients that are used to optimise neural network weights (LeCun et al. 1998). The loss, i.e. the error, is calculated in the neural networks output and distributed back towards the inputs in order to update the weights. (Dreyfus 2005) Detailed description of the backpropagation algorithm can be found for example in (Dreyfus 2005) and (Haykin 2009).

Weights of a neural network must be initialised before the network training with gradient descent based algorithm can be started. Initialisation of the weights has a great impact on the resulting model. When made properly, the training converges quickly. (Cui and Fearn 2018) When the weights are initialised randomly, the experiments should be performed multiple times to ensure that the result is reliable. (Graves 2012) In this study, the weights are initialised randomly in Keras by drawing values randomly from either uniform distribution or normal distribution. Default parameters that define the distributions in Keras are used, which are mean of zero and standard deviation of 0.05 for normal distribution and range -0.05–0.05 for uniform distribution. (Anon 2019c)

In this study, a backpropagation-based algorithm called Adaptive moment algorithm (Adam) is used in the neural network training. Adam was introduced by (Kingma and Ba 2014) and it is a stochastic gradient descent method. Instead of using fixed learning rate, Adam calculates learning rates for each weight of the network adaptively. (Kingma and Ba 2014). Other gradient descent based weight optimisation algorithms included in many neural network libraries are presented for example in (Ruder 2016).

3.4.2 Regularisation

Overfitting of a neural network on training set can be prevented or decreased by using one or more regularization methods. L2 is a common regularisation method which helps the network to use all the weights of the network so that each contributes useful information regarding the behaviour of the modelled system. It is based on penalising the weights' sum of squared amplitude. (Cui and Fearn 2018) In L1 regularisation, a maximum value for the sum of neural network weights is set which affects the learning

capacity of the network thus reducing overfitting. The penalty term in L1 and L2 methods is applied to the loss function.

Dropout is another regularisation method, in which neurons in the neural network are turned off during an epoch in the network training phase when the weights of these neurons are not updated on that epoch. Practically it has the same effect as applying noise to the signals going to the units (Srivastava et al. 2014) Applying noise to the model inputs can also be used in regularisation. In Keras, e.g. dropout can be applied on layer and by defining dropout rate to e.g. 0.2, then 20 % of randomly selected input units to that layer are set to 0. (Anon 2019d) Max-norm regularisation specifies a maximum length for the vector of weights that are connected to each hidden unit in a neural network layer. In practice, the methods constraints the Euclidian norm *p* given by: (Anon 2019e)

$$p = \sqrt{w_1^2 + \cdots + w_n^2}, \tag{4}$$

where *p* is the norm,
w is the value of the weight and
n is the number of connections

For example, consider a neural network with four neurons in the input layer and one in the hidden layer and therefore four connections in total to the hidden neuron. Applying the max-norm regularisation with value 2 constraints maximum value of *p* in equation 4 to 2, i.e. the maximum value that the weights can get is constrained. Dropout and max-norm were found to regularise neural networks well when applied together. (Srivastava et al. 2014) Both of these were included in the hyperparameter search during the case study 2 of this research. In the first case study, no regularisation methods was applied.

3.5 Hyperparameter optimisation

Hyperparameters, such as the number of hidden neurons in a hidden layer and the number of hidden layers, defines the size of a neural network. Other hyperparameters like the optimiser and its learning rate have an effect on the training process. Purpose of hyperparameter optimisation is to find hyperparameters that provide good model performance and thus is an essential phase in neural network modelling. Tuning hyperparameters manually is time consuming even when performed by an expert.

Multiple different hyperparameter optimization strategies exist. Grid search is an exhaustive brute-force method for hyperparameter optimisation. In grid search, limits and step size in the grid for numerical hyperparameters and options for non-numerical hyperparameters are defined (see Table 2). (Bergstra and Bengio 2012)

Table 2. Example of a search grid specified for three model hyperparameters.

Hyperparameter	Low limit	High limit	Step size in grid	Possible values/ options
Number of hidden layers	1	5	1	1,2,3,4,5
Number of hidden neurons	4	32	4	4,8,12,16,20,24,28,32
Optimisation algorithm	-	-	-	‘adam', 'sgd'

The total number of different hyperparameter combinations is calculated by multiplying the numbers of possible values of each hyperparameter. In the example in Table 2, there are five possible values for number of layers, eight possible values for number of neurons and two possible options for optimisation algorithm. Thus, the number of different models to be created would be $5 \times 8 \times 2 = 80$. Including more hyperparameters and making the step size smaller in the grid search lead to so called curse of dimensionality as the amount of possible hyperparameter combinations increase exponentially. The number of hyperparameters that have significant effect on the neural network performance varies between different datasets and applications which suggests that grid search is not an optimal way for hyperparameter tuning. (Bergstra and Bengio 2012)

Random search is another brute-force optimisation method in which the hyperparameters of the neural network are drawn randomly from a specified search space to be used in training. Bergstra and Bengio (2012) compared random search and grid search in HPO of single-layer network, proving that random search is more efficient than grid search if given the same computational resources and the hyperparameters are chosen from the same search space. Their studies suggest that this is due to differences in significances of hyperparameters. (Bergstra and Bengio 2012).

Neither grid nor random search make use of performance results of already trained models. Sequential, adaptive search methods address this by directing the search in promising search space by taking previous results into account. (Bergstra and Bengio 2012). This kind of directed search algorithms are also referred to as informed search

algorithms in the literature due to their nature of taking the results of previously tested hyperparameter combinations into account when deciding which are tried next. Gaussian Process (GP) and Tree-structured Parzen Estimator (TPE) are two sequential model-based search methods that may be useful when the actual objective function is computationally too heavy, as they approximate the objective function with a lighter surrogate. (Bergstra et al. 2011) TPE algorithm was tested in the second case study but it did not provide more accurate models than random search, thus it was no further experimented with in this research.

Also various other algorithm-based hyperparameter optimisation strategies for automatic tuning have been developed to find optimal hyperparameters efficiently, i.e. to find global optimum solution with least amount of computational resources. Differential evolution algorithm is proposed in (Nakisa et al. 2018). Also optimisation methods such as genetic algorithm, particle swarm optimisation, simulated annealing and tree of Parzen estimators have been used in hyperparameter optimisation. In context of deep CNN networks, Hinz et al. (2018) suggest that optimal hyperparameters for a model trained on specific image dataset can be found by using lower resolution images at first and then the higher resolution images are used in training.

3.6 Model selection

Input variable selection and hyperparameter optimisation results in many candidate models which need to be evaluated to find the best one. When dealing with data-driven models, the model needs to be able to mimic the relationship between input and output variables with high enough accuracy and to generalise well at the same time. In practice, a compromise has to be made with these two targets because training a model that learns the whole training dataset with all the noise, is not a model that is able to produce accurate predictions with unseen data, i.e. data that is different from the training data. In model selection, the models that are overfitting are rejected and the generalisation errors are estimated to find the lowest one. (Dreyfus 2005, p. 131 & 133)

Cross-validation is one method to estimate model generalisation. In cross-validation, the training dataset is split in K parts which are about the same size and one of these parts is used for validation whilst the others are used for training. This procedure is repeated K

times so that each of the *K* parts have been used in validation (see Figure 8). (Dreyfus 2005, p. 135)

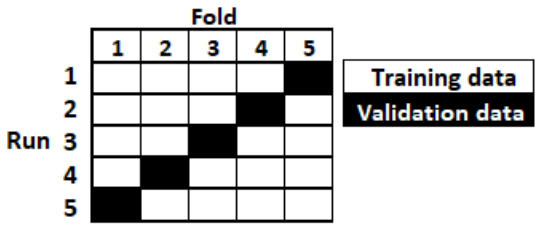


Figure 8. Principle of data division in k-fold cross-validation with $K = 5$ as an example.

In (Dreyfus 2005, p. 135), it is mentioned that multiple models should be created with each dataset combination by using different initial conditions. With neural networks, this means e.g. initialising the network with different weights. Mean squared error on the validation is calculated for each model with different initial values and the lowest one is saved. After doing this for each *K* dataset combinations, a cross-validation score is calculated. Statistically, the cross-validation score is a better estimate of the generalisation error than a score that is calculated by using only one static, e.g. 20 % share of the dataset, in validation. (Dreyfus 2005, p. 135) The division of the dataset into *K* parts is usually made pseudo-randomly and thus, the process needs to be repeated multiple times. (Baumann 2003)

After good hyperparameters are found by measuring the model performance on the validation dataset, the model needs to be tested against an independent testing dataset. This is required to see whether the model work on previously unseen data and thus shows if the generalisation ability of the model is good. This is because when using the validation dataset to fine tune the hyperparameters by creating maybe thousands of models, the estimation of the model generalisation ability using the validation dataset becomes unreliable. (Ng 2009) In this study, separate training, validation and testing datasets are used. The validation dataset is used to test each of the generated model in multi-step-ahead configuration because one-step-ahead results were noticed not to be a reliable measure on the model performance, i.e. generalisation. Normalised root mean square error (NRMSE) value for the validation dataset is used to rank the models. The models of each model type are then tested using the independent testing dataset to get more reliable measure on the model generalisation ability.

4 DYNAMIC ARTIFICIAL NEURAL NETWORK ARCHITECTURES

Depending on the application, the model should be able to either map the input-output relationships between variables, i.e. produce one-step-ahead predictions, or produce multi-step-ahead predictions. Multi-step-ahead predicting is within the scope of this study, as it is a baseline requirement for the data-driven model to produce new predictions in Apros on each simulation time step. A machine learning model that is producing multi-step-ahead predictions can be also called a simulator (Dreyfus 2005). In this chapter, autoregressive models that lay foundation for dynamic modelling are presented and then multiple recurrent artificial neural network architectures are presented. Many other neural network architectures exist in addition to the ones presented in this study. These include for example convolutional neural networks (CNN) that are commonly used in image recognition tasks. Although CNNs are not considered in this study, CNN models have been used in sequence modelling for example in (Bai et al. 2018). The authors present a generic temporal convolutional network (TCN) and show that it can outperform e.g. LSTM network in many sequential modelling problems. They mention that in addition to being more accurate, the TCN model is also easier to interpret.

Different techniques for making multi-step-ahead predictions have been used. In indirect techniques, one-step-ahead model is used recursively to produce predictions for the next time step using the predictions from the previous time step. Thus the indirect methods are often also referred to as recursive strategies. The model may also include the so called exogenous inputs which is the case in this study. Direct techniques include so called prediction horizon which defines how many time steps ahead the model produces predictions at once. The recursive methods mentioned above may be inaccurate in problems with a long horizon because the models are trained to produce accurate one-step-ahead predictions. In addition to that, the prediction errors at previous time steps cumulate to the next ones. (Taieb et al. 2010) & (Taieb 2014). According to (Taieb 2014), determining if recursive or direct method is the most suitable, needs to be done empirically for each case separately.

Artificial neural networks have been used in creating static and dynamic models from simulator data for example in (Li et al. 2015). Li et al. (2015) modelled post-combustion

CO_2 capture process using simulated data and performed multi-step-ahead predictions with the indirect approach. In their case study, the model was able to produce accurate multi-step-ahead predictions for about 90 steps ahead, which the authors comment to be enough for model predictive control and online optimisation tasks. (Li et al. 2015)

4.1 Autoregressive models

Autoregressive (AR) model as a model class is defined by having its own output values from previous time steps as regressors in calculating the next output. The number of lagged values of the output variable can be specified and the lagged values do not need to be successive ones. For example, previous values from time steps k - 1, k - 3, and k - 5 could be used instead of k - 1, k - 2 and k - 3. The most simple autoregressive model is AR (see Equation 5) that does not include any other inputs but only the output values from previous time steps: (Aguirre and Letellier 2009)

$$y(k) = a_1 y(k-1) + \cdots + a_{n_y} y\left(k - n_y\right), \tag{5}$$

where $y(k)$ is the output to be predicted,
$y(k\text{-}n_y)$ is the output from the previous time step(s),
a_{n_y} is a parameter (or regressor coefficient) and
n_y is the maximum number of output lags used.

ARX (autoregressive model with exogenous inputs) model includes the same components as AR model but adds a so called exogenous input, or the system input, to the right hand side of the (5). The structure of an ARX model is: (Dreyfus 2005) & (Aguirre and Letellier 2009)

$$\begin{aligned} y(k) = a_1 y(k-1) + \cdots + a_{n_y} y\left(k - n_y\right) + \\ b_1 u(k-1) + \cdots + b_{n_u} u(k - n_u), \end{aligned} \tag{6}$$

where b_{n_u} are the parameters for the exogenous input regressors and
n_u is the maximum number of input lags used.

A NARX (Nonlinear ARX) model is based on ARX model but the right-hand side of (5) is a nonlinear function. Thus the NARX model can be used to model nonlinear relationships between the input and output variables with their lagged values.

4.2 Artificial neural networks

The idea of artificial neural networks is to mimic human brains and its way of processing information. Artificial neural networks are data-driven models that are trained to do specific computational tasks. (Haykin 2009, p. 2) Artificial neural network models can be divided in two broad classes: regression models and classification models. In regression modelling, the target is to find a model that explains the behaviour of one or more response variables, using values of the regressor variables. In classification problems, the target is to classify a given input vector into one discrete class. Classical example is to take an image and classify whether there is a dog or a cat in it. (Bishop 2006) An industrial example of a classification problem is identifying different faults occurring in a process plant. In this study, the task is to use synthetic simulator data to model dynamic chemical processes. Regression type of the artificial neural networks are applied in making multi-time-step predictions on how the process behaves.

Artificial neural networks can be built using different architectures that are suitable for different kinds of applications. One of the most traditional artificial neural network architectures is multilayer perceptron (MLP) shown in Figure 9. The multilayer perceptron is a feed-forward network which means that neurons' output values are not fed back locally or globally (Haykin 2009, p. 124). Globally recurrent linear ARX and NARX network architectures are presented in Subsections 4.2.1 and 4.2.2 and Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) architectures that include local feedback loops are presented in Subsections 4.2.3 and 4.2.4, respectively. In the case studies of this research, each of the model types are built in a way that they include exogenous inputs and autoregressive inputs.

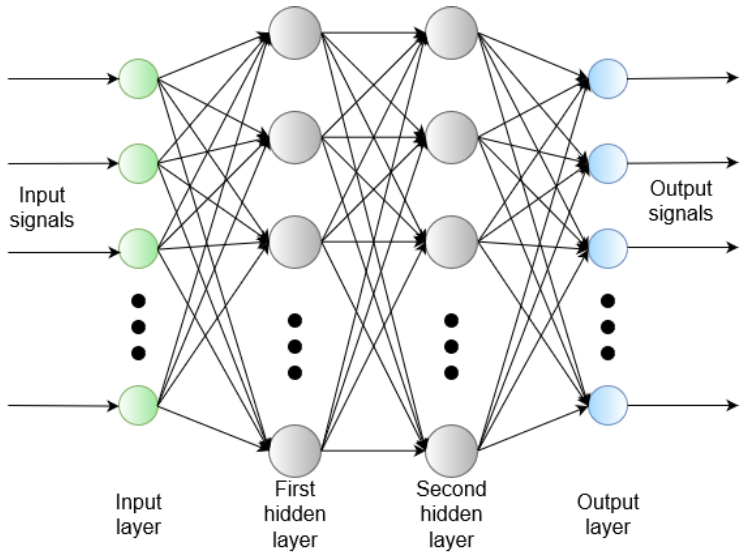


Figure 9. Multilayer perceptron with an input layer, two hidden layers and an output layer. Adapted from (Haykin, 2009 p. 124).

The first layer of the neural network is called the input layer where the input values are fed. Neurons in the input layer (green neurons in Figure 9) do not include calculations but they connect each of the input signals to each neuron in the following layer. The last layer is called the output layer which neurons output (blue neurons in Figure 9) the values that the network has calculated. In between, there is one or more hidden layers which neurons are marked in grey in Figure 9. Each of the neurons in a single layer of the network is connected to each of the neurons in preceding and subsequent layers.

Hidden neurons and output neurons of an artificial neural network are simple computational units that together form the network. Structure of a single hidden neuron is shown in Figure 10. The output of this neuron is the function of input signals values, weights and bias. Depending on the chosen activation function, the output is linear or nonlinear. (Haykin 2009) Neuron output y in a case of linear activation function is in a mathematical for:

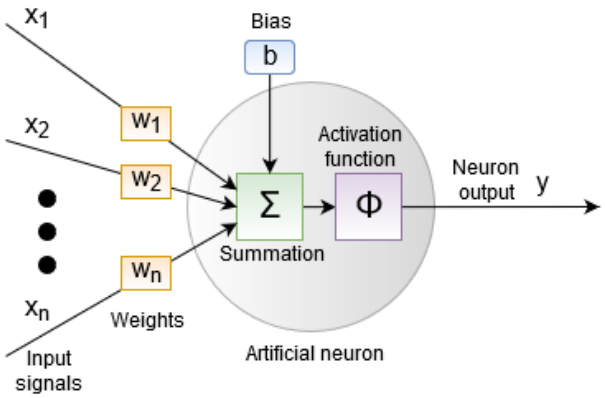


Figure 10. Structure of hidden and output neurons in an artificial neural network. Adapted from (Haykin 2009, p. 12).

$$y = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + b, \tag{7}$$

where y is the neuron output,
$x_1, x_2, \ldots, x_n$ are the values of the neurons input signals,
$w_1, w_2, \ldots, w_n$ are the weights of the respective input signals and
b is the bias term.

The number of hidden layers and neurons, the connections between them and activation functions are hyperparameters of a neural network. The more layers and neurons network have, the more complex considering the number of mathematical operations. This also increases the mapping capability of the network, meaning that more complex functions can be approximated. Activation functions commonly used in neural networks include e.g. linear function, sigmoid function and hyperbolic tangent function (Haykin 2009, p. 14).

4.2.1 Autoregressive network with exogenous inputs

Autoregressive neural network model with exogenous inputs (ARX) is a linear model structure that can be used in dynamic modelling. Compared to NARX model, the only difference is that NARX include nonlinear elements such as nonlinear activation function in a neural network. In this study, linear ARX neural network model is developed in both case studies to act as a benchmark model for the nonlinear neural network types to see if there is differences in the accuracy of the models. The simplicity of ARX model structure (see Figure 11) makes it easy to train in neural network form which makes it a preferable choice if there are no nonlinearities in the modelled dynamics. (Haykin 2009, p. 792-793)

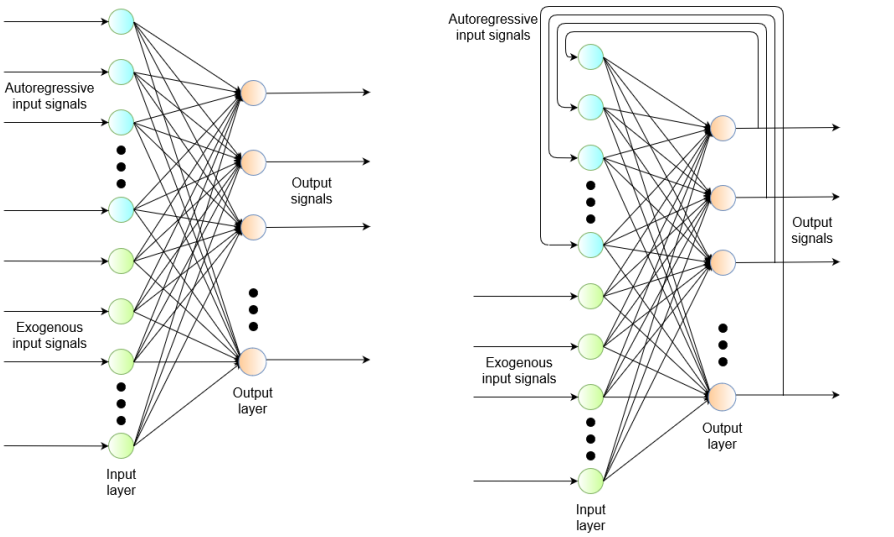


Figure 11. Structure of the ARX neural network in open-loop and closed-loop configurations. Adapted from (Haykin 2009, p. 792).

4.2.2 Nonlinear autoregressive network with exogenous inputs

Neural networks can include local or global feedback loops. Feedback connection is local when it is applied to only one neuron, although there may be multiple neurons that has local feedback. In recurrent neural networks, outputs of one or more layers and/or of the whole network are fed back as input. Nonlinear dynamic model is achieved when recurrent neural network architecture that contain nonlinear activation functions is combined with feedback loops that contain unit-time delay components. (Haykin 2009, p. 23 & 836) Nonlinear autoregressive neural network with exogenous inputs (NARX) is an architecture which can be used for multi-step-ahead predicting by feeding the output signals back as inputs, forming a closed-loop NARX neural network architecture shown in Figure 12. The teacher forcing training strategy introduced in the previous chapter can be applied in training the NARX neural network alike with the ARX neural network. The network configuration in that case is as shown in Figure 13.

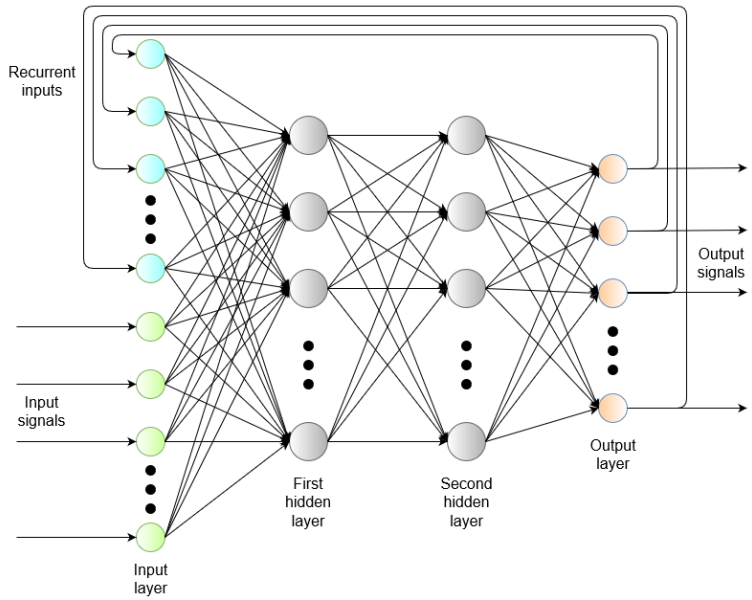


Figure 12. NARX neural network in closed-loop configuration, showing how the output signals are connected to the input layer of the network. Adapted from (Haykin 2009, p. 792).

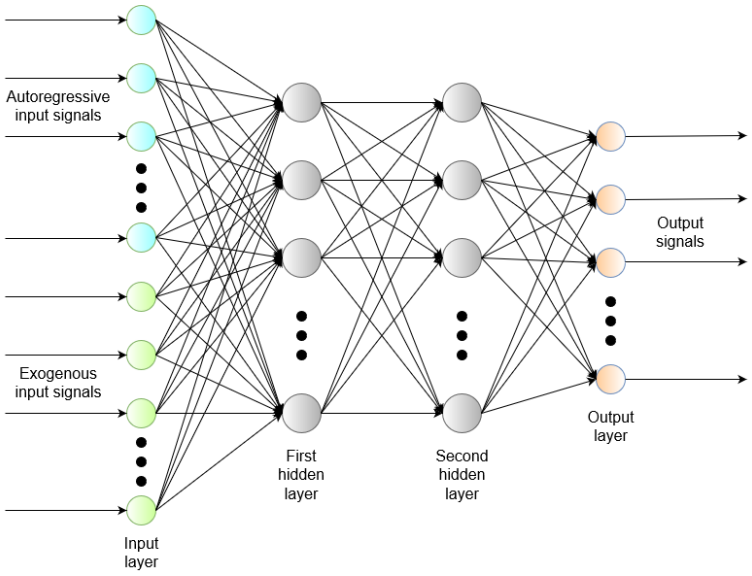


Figure 13. NARX neural network in open-loop configuration that is used during the teacher forcing training. (Haykin, 2009, p. 817).

NARX neural network has been applied in modelling nonlinear behaviour of reactor-exchanger process in (Chetouani 2008). The NARX neural network have been also used

in multi-step-ahead predicting of water level in a floodwater storage pond in (Chang et al. 2014), and in dynamic modelling of braking torque behaviour in (Ćirović et al. 2012).

4.2.3 Long Short-Term Memory network

Inputs and outputs of real nonlinear dynamic systems may include short-term or long-term dependencies, or both (Haykin, 2009, p. 818–819). Traditional recurrent neural networks (see Figure 14) are unable to address long-term dependencies when predicting due to vanishing gradient problem that occurs with traditional back-propagation through time (BPTT) and real-time recurrent learning (RTRL) training methods. It means that during the training, the value of error signal going backwards in the network vanishes or grows exponentially depending on the activation functions. When the error signal is vanishing, the learning of long-term dependencies is prevented or takes excessively long. Exponentially growing error signal may cause weights of the network to oscillate. (Hochreiter and Schmidhuber 1997).

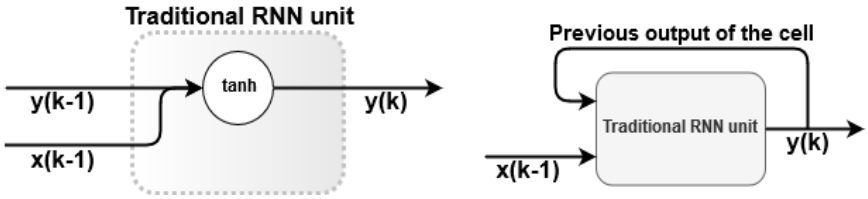


Figure 14. Structure of a single unit of a traditional recurrent neural network unit which includes a local feedback loop.

Long Short-Term Memory (LSTM) is a type of recurrent neural network architecture that was introduced in 1997 to address the vanishing gradients problem. According to developers, the network is able to learn long-term dependencies up to 1000 time steps. (Hochreiter and Schmidhuber 1997) The LSTM network is built of LSTM units, i.e. cells, with input and output gates. One variant of LSTM adds a forget gate as shown in Figure 15. The forget gate gives the memory cell an ability to reset itself that in practice means forgetting irrelevant previous inputs. (Graves 2012, p. 38 & 42) Thus, the network is able to produce good results even when the network is fed with inputs for endless amount of time steps (Gers and Schmidhuber 2001). The LSTM units have a built-in recurrent connection with a Constant Error Carousel (CEC) unit that defines the state of that cell, S_c, in form of an activation value. The CEC unit is the component that solves the vanishing

gradient problem as it prevents error flow from vanishing. The input gate unit prevents memory content of the LSTM unit from being affected by currently irrelevant inputs. Likewise, the output gate unit prevents other LSTM units from being affected by currently irrelevant elements in the memory content. (Gers and Schmidhuber 2000b) & (Hochreiter and Schmidhuber 1997) This means that e.g. when the input gate has an activation of 0 and thus is closed, the input will not affect the activation value of the cell. Therefore, the information stored in form of cells activation value remains and can be taken into use later by opening the output gate. The next state of an LSTM cell is given by: (Gers and Schmidhuber 2000a)

$$S_c(\mathrm{k}) = S_c(\mathrm{k}-1) \times y_f + y_{in} \times g, \tag{8}$$

where S_c is the cell state,

y_f is the forget gate activation value,

y_{in} is the input gate activation value and

g is the cell input signal squashed through hyperbolic tangent function.

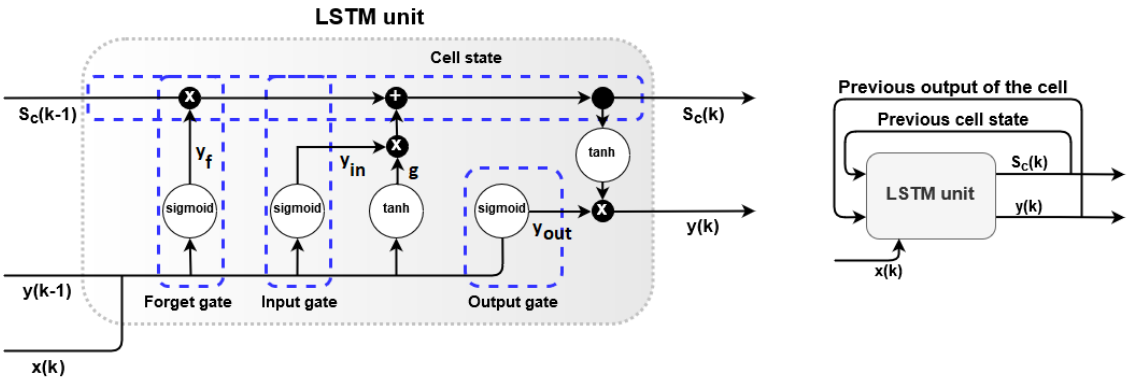


Figure 15. LSTM unit with input, output and forget gates, showing the connections in the unit. Adapted from (Gers and Schmidhuber 2000b).

The input, output and forget gates include sigmoid functions that outputs values in range -1–1. The signals connected to the gates have each their own weights W which are not shown in (8) and (9) (see equations (10)–(12) in Subsection 4.2.4 for example of how the signals are weighted). Other activation functions are in the unit are hyperbolic tangent functions, outputting values in range 0–1. The output of an LSTM cell is defined by: (Gers and Schmidhuber 2000a)

$$y(\mathrm{k}) = tanh\big(S_c(k)\big) \times y_{out}, \quad (9)$$

where y is the cell output,
S_c is the cell state and
y_{out} is the output gate activation value.

Another LSTM variant adds peephole connections to the original architecture. The peephole connections are weighted connections from the previously mentioned CEC to each gate of the unit that are not shown in Figure 15. During the training, the errors are not backpropagated through the peepholes towards CEC. Through peepholes on the other hand, the gates get information about the state of the memory cell (Gers and Schmidhuber 2000a). In this study, Keras Python library is used which LSTM network implementation does not include peephole connections.

LSTM networks has been used in various applications such as speech and handwriting recognition that are cases in which the long-term memory is valuable (Graves 2012, p. 41). LSTM network was used in multi-step-ahead predicting of wind speed in (Wang et al. 2018). LSTM, traditional RNN and GRU that is presented in the next section, were compared in predicting remaining useful life of an aircraft turbofan in (Wu et al. 2018), concluding that LSTM performed better in each studied case. In (Yuan et al. 2019), a modified LSTM called Supervised LSTM (SLSTM) was used in developing nonlinear dynamic soft sensor model of debutanizer column unit operation and penicillin fermentation process, in which it performed better than LSTM and traditional RNN. In this study, each of the studied models include exogenous inputs in addition to autoregressive ones. This is also the case in (Guo and Lin 2017), where LSTM neural network was compared to many alternatives on multiple datasets, showing that the LSTM network was more accurate in making predictions and also easier to interpret.

4.2.4 Gated Recurrent Unit network

Gated Recurrent Unit (GRU) neural network architecture was introduced in (Cho et al. 2014). GRU network share similarities with LSTM neural network but has a simpler structure. It includes components that make it also capable for remembering longer dependencies. The GRU cell do not include separate output signal *y(k)* and the hidden

state signal *h(k)* like the LSTM cell but instead, *y(k)* = *h(k)*. There are two gates in the Gated Recurrent Unit which are the update gate and the reset gate (see Figure 16).

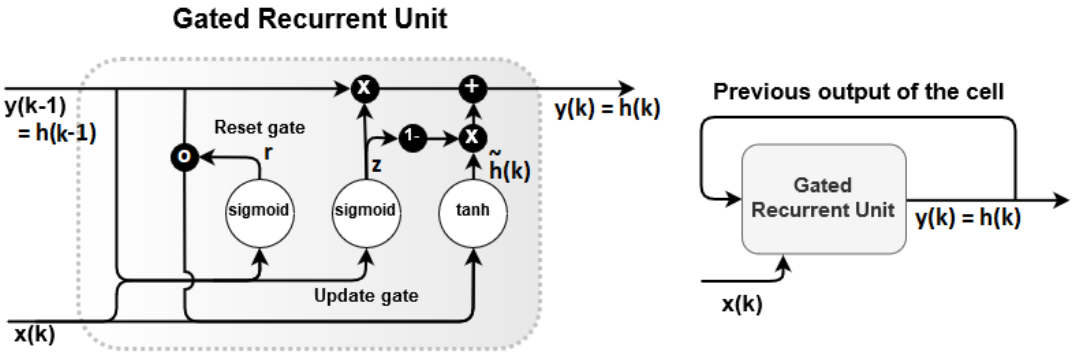


Figure 16. Structure of a Gated Recurrent Unit, showing the reset and update gates. (Cho et al. 2014).

The update gate controls what information from the GRU cells state from the previous time steps is used to calculate the next state of the cell, i.e. how much the activation of the unit is updated. The reset gate on the other hand allows the control of forgetting the previously computed state. (Chung et al. 2014) Activation of the reset gate of a single Gated Recurrent Unit can be expressed as: (Cho et al. 2014)

$$r = \ sigmoid([W_r x(k)] + [U_r h(k-1)]), \tag{10}$$

where *r* is the reset gate activation
W_r and U_r are weight matrices (i.e. parameters which are learned),
x(k) is the input signal to the unit and
h(k-1) is the the previous hidden state of the unit.

Activation of the update gate is given by:

$$z = \ sigmoid([W_z x(k)] + [U_z h(k-1)]), \tag{11}$$

where *z* is the update gate activation
W_z and U_z are weight matrices and
x(k) and *h(k-1)* as in (9).

So called candidate activation $\tilde{h}$ of the cell is given by:

$$\tilde{h}(\mathrm{k}) = tanh([Wx(k)] + [U \times (r \circ h(k-1))]), \quad (12)$$

where $\tilde{h}(\mathrm{k})$ is the candidate activation of the cell,
W and *U* are weight matrices,
x(k) and *h(k-1)* as in (9) and (10), and
∘ denotes Hadamard product.

Now, the activation of the GRU cell, $h(k)$, can expressed as: (Cho et al. 2014)

$$h(k) = z \times h(k-1) + (1-z) \times \tilde{h}(k). \quad (13)$$

Traditional RNN with hyperbolic tangent activation function, LSTM and GRU neural networks were compared in music and speech signal data sequence modelling in (Chung et al. 2014), concluding that both LSTM and GRU were better than the traditional RNN.

4.3 Modular and ensemble models

With very complex systems, it might be difficult to build a single model that can describe the behaviour of this system. One solution is to build a modular model that contain multiple so called local models, each responsible for some specific operation area. (Abrahart et al. 2008) For example, slow and fast dynamics of the system could be modelled with separate models or each of the output variables could have their own dedicated models. Ensembling methods are techniques that are used in machine learning to reduce generalisation error by using multiple models. One ensemble method is bagging, where multiple models are trained individually and a voting mechanism in the final ensemble model determines the model's output. Other ensembling methods use model averaging that is also the case in the bagging method. Multiple different models most likely do not produce the same mistakes which can make model averaging valuable. (Goodfellow et al. 2016) & (Abrahart et al. 2008) Bagging neural network models have been built in (Wu and Peng 2017) for the same purpose in predicting the amount of power generated by a wind turbine.

Boosting is another ensemble modelling method that makes use of multiple models to build up a better predictor than an individual model can. The approach in boosting is to divide the original training dataset into multiple datasets and use each to train separate models. The prediction, that this kind of boosting ensemble model produces, is e.g. a weighted average of the separate models' outputs. Boosting has been used with neural networks for example in (Bian et al. 2018), showing that it can improve stability and accuracy of the neural network model.

Hybrid models that combine multiple different model types can also be classified as modular models. One hybrid neural network approach for time series predicting is proposed in (Xu et al. 2019). In their approach, the hybrid model consists of linear autoregressive model or autoregressive integrated moving average (ARIMA) model combined with a nonlinear deep belief network (DBN). The idea in the approach is that first, a linear model is fitted to the data and then the residuals of the linear model are modelled with nonlinear DBN. Summing up the prediction of the linear model and the DBN forms the actual prediction. (Xu et al. 2019)

5 CASE STUDIES

Data-driven modelling with artificial neural networks was discussed in the previous chapters. In this chapter, the general workflow of creating a data-driven surrogate model of a physics-based unit operation model in Apros is presented. Also, the steps to implement the surrogate model back in Apros are defined. This research included two case studies. The first one includes a simple physics-based model of a water tank which liquid level is controlled. The second case study model is a methanation reactor in a power-to-gas process. The case specific details are presented in Sections 5.2 and 5.3.

5.1 Workflow and modelling framework

The overall workflow presented in Figure 17, starts from having a dynamic physics-based Apros model from which a unit operation model is selected to be modelled with data-driven methods. The process continues by selecting input and output variables that are relevant to the selected unit operation.

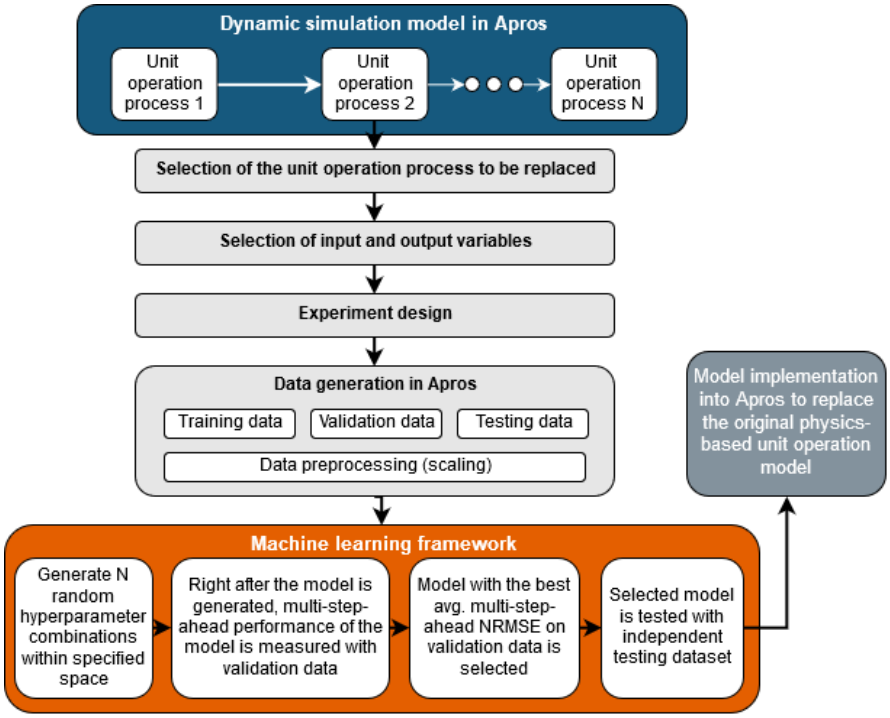


Figure 17. Overall workflow to create a surrogate model of a unit operation process in Apros showing the steps from selecting the unit operation model to implementing the machine learning model in Apros.

To generate data for data-driven modelling, an experiment design is needed. According to the selected experiment design, experiments in Apros are run and the values of previously selected input and output variables are logged in a text file. Data is generated separately for training, validation and testing purposes. The machine learning framework built for this study is based on building neural networks with Keras (Chollet 2015). Before the data is imported into the framework, it is pre-processed by scaling the variables values in the same scale. The framework is presented in more detail in Subsection 5.1.3.

5.1.1 Selecting input and output variables

The selection step includes choosing Apros modules which will be included in the surrogate and which will act as an interface between the physics-based model and the data-driven model. As a consequence, input and output variables of the data-driven model are defined. For this step, it is necessary to understand how different Apros modules can be configured so that they use the outputs of the data-driven model as inputs, instead of the Apros solver calculating them. In practice this means setting up component properties in a way that allows values of properties like enthalpy or pressure to be set on each simulation time step. This is looked into in more detail in the case studies. After the input and output variables for the model have been selected, the process to produce dataset for data-driven modelling continues as shown in Figure 17.

5.1.2 Experiment design and dataset generation

After the input and output variables have been selected for the data-driven model, the next step is to make a design of experiments which in practice is a sequence of changes in variables that cause either A) direct or B) indirect excitation in the input variables of the system (see Figure 18). Direct excitation means that if e.g. pressure in Apros point module PO03 is the selected input of the data-driven model, then the pressure value of PO03 is changed directly during experimentation which causes response in the output variables. Causing indirect excitation to PO03 pressure can be made for example by adjusting opening of a control valve located before point PO03 that leads to change in PO03 pressure as the pressure loss over the control valve changes.

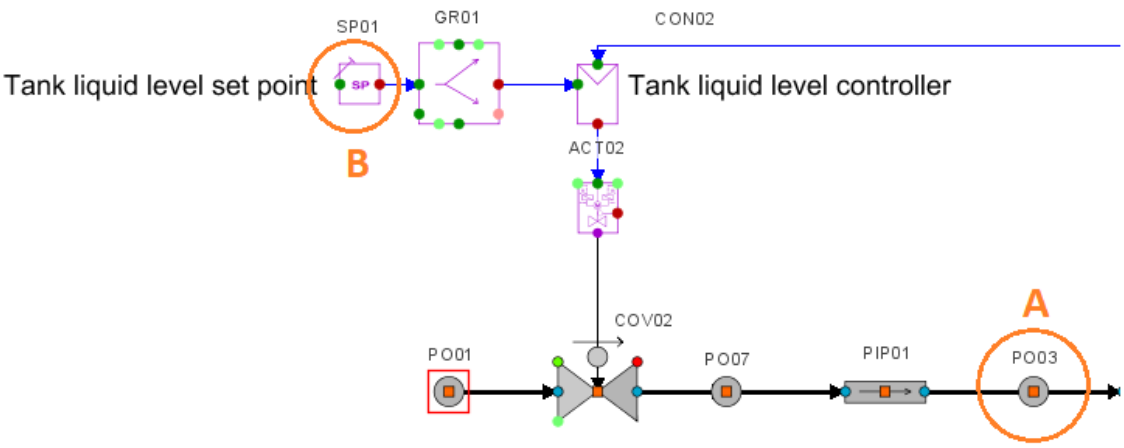


Figure 18. Alternatives for causing excitation to an input (pressure of PO03) of a data-driven model. A) Changing pressure directly. B) Changing controller setpoint value that indirectly cause change in pressure by adjusting pressure loss over control valve COV2.

In the first case study of this research, a tailor-made experiment design was used as the model included only one excitation variable. In the second case study, the inscribed type of central composite experiment design presented in Subsection 3.1.1 was chosen for the generation of neural network training data. The experiment designs are shown in Sections 5.2 and 5.3.

Having input and output variables selected and experiment design ready, the dataset is generated by running the experiments in Apros as shown in Figure 19. After this, data logging in Apros is set up so that input and output variables are logged in separate files. In both of the case studies, the sampling frequency was set to 0.2 s in Apros, which is the same as the simulation time step. Input and output data files from Apros are combined into a single CSV-file in which headers are created to show which variable is input and which is output (by “I” and “O”). Also, the description of the variable is marked in the header. This procedure is repeated three times to generate training, validation and testing datasets.

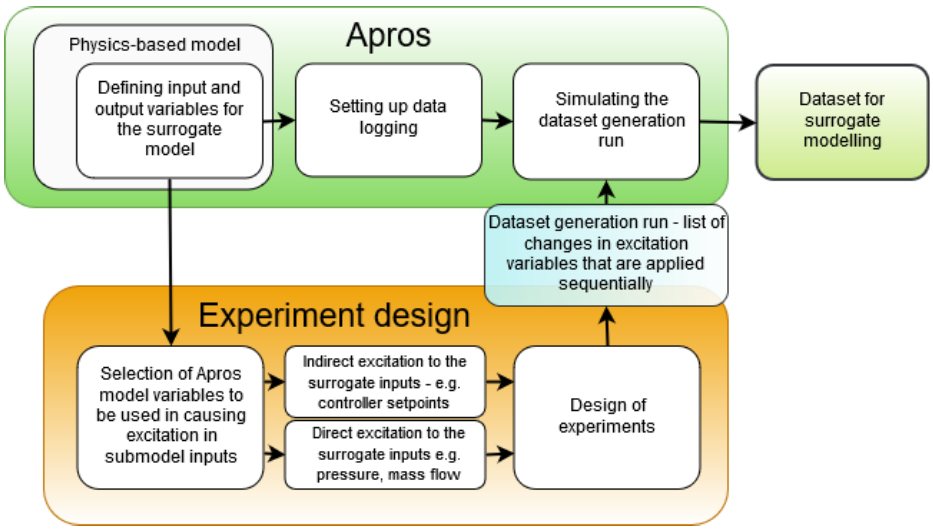


Figure 19. The workflow to generate dataset in Apros for surrogate modelling.

5.1.3 Machine learning framework

The framework built for this study is based on neural network modelling using Keras which is an open source Python library and a neural network API (Application Programming Interface) that utilises TensorFlow as the backend. Keras is designed for quick development and testing of neural networks and is widely used in research. It is able to use both CPU and GPU in performing calculations. (Chollet 2015) TensorFlow is an open source software library that can be used for machine learning and deep learning (Anon 2015). Calculations in this study are performed on a laptop CPU (i5-7300U, 8 GB RAM) and on a GPU-server with Nvidia Tesla P100 GPU.

The framework allows the user to build linear ARX, NARX, LSTM and GRU neural networks using Keras. The basic structure of the algorithms for each is the same. First, the user specifies files that contain the training and the validation datasets. The inputs and outputs in the datasets are marked by “I” and “O” respectively, as mentioned in the previous section. In the modelling framework, this allows the user to select individual input and output variables that are to be included in the model. The user can also choose the number of lagged values for input and output variables. For example, if the chosen number of input lags is three, then three lagged values of each selected input variable are included in the model as additional inputs.

Next step in the framework is specifying the hyperparameter space that is to be used in the random hyperparameter optimisation process. The number of neural network layers

and their hidden neurons, hidden layers' activation functions, output activation functions and weight initialisation method can be chosen. Adam weight optimiser was used in this study and the framework allows to choose its parameters which are the learning rate (i.e. step size), learning rate decay and exponential decay rates β_1 and β_2 for moment estimates that the algorithm calculates. Also the range for the number of training epochs and the batch size can be specified. The training samples are shuffled during training.

Multiple strategies for applying dropout regularisation to the network are provided for the hyperparameter optimisation. Hyperparameter *n_dropout* is set to 0 if no dropout is applied, to 1 if dropout is applied to the input layer of the network, to 2 if it is applied to the output layer and to 3 if it is applied to the input and output layers. *Dropout_rate* specifies the fraction of the input units that are set to 0 randomly during the training process. *Maxnorm* hyperparameter specifies a maximum value to the norm of each hidden unit's weights. These regularisation methods were introduced shortly in Subsection 3.4.2.

Based on the user specified hyperparameter space and the number of hyperparameter combinations, N, to be generated, the algorithm generates the combinations pseudo-randomly in a list. After that, the algorithm loads the training and validation datasets and modifies them according to which input and output variables the user chose previously and how many input and output variables are to be included in the model. The input data is then reshaped from two to three-dimensional array if LSTM or GRU neural networks are chosen as the layers in these networks require three dimensional input shape. The dimensions in the 3D-array represent batch size, time steps and input dimension (i.e. number of input variables) as shown in Figure 20. The batch size specifies how many input samples are used in the training before the weights are updated. For example in case of 1 000 input samples and batch size of 100, one epoch of training contain 10 batches. Time steps specify how many previous time steps are included in the model.

After the input has been reshaped, the algorithm goes into the hyperparameter optimisation loop, in which neural network models are trained using the previously generated hyperparameter combinations. Inside the loop and right after the model has been trained, the models are tested using the validation dataset in closed-loop, i.e. multi-step-ahead, configuration as shown in Figure 12 in Subsection 4.2.2. This is because one-step-ahead performance was noticed to be uncorrelated with the multi-step-ahead performance, i.e. the validation loss that Keras calculated was not a reliable measure for

goodness of the model. Example of one-step-ahead and multi-step-ahead performance of 800 linear ARX models can be seen in Chapter 6.

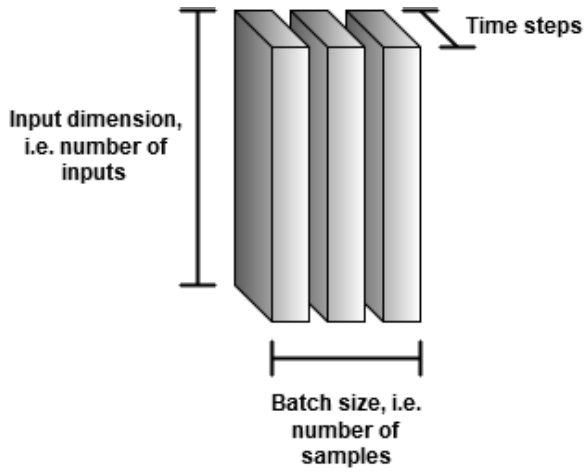


Figure 20. The axes of three-dimensional input data that the recurrent layers in Keras require.

The predicted values from each time step in multi-step-ahead test bench are stored and root mean square error (RMSE) value for each output variable is calculated. To make the error values between variables and datasets comparable, the RMSE values are normalised which gives the NRMSE values. In addition, the average of the outputs NRMSE values is stored. These NRMSE values are written in a data file together with the hyperparameters that were used to obtain the results, and the data file is updated as the hyperparameter optimisation loop continues. The algorithm also prints the NRMSE values of each model and output variable after testing and also the current best NRMSE values. For the model, the current best value is the average over all the outputs. This way the user can monitor the hyperparameter optimisation process and stop it if a model with high enough performance has been found before all N hyperparameter combinations have been gone through. The algorithm was also set to save each model that achieved an average NRMSE value lower than 2 % on the validation dataset in the multi-step-ahead test bench. The model with the best average NRMSE value is then tested in a separate test bench that allows to use of any dataset for testing the model in multi-step-ahead configuration.

5.1.4 Implementing machine learning model into Apros

The machine learning model created using Keras can be implemented in Apros by using a User Component module and attaching two SCL scripts and a SCL module in it. The SCL module contains two strings that include multiple lines of Python commands. These strings are from now on called 1) initialisation statement and 2) prediction statement in this study. The SCL scripts are executed during different phases of the simulation and make use of the statements defined in the above-mentioned SCL module. As shown in Figure 21, the initialisation statement is executed during preparation of the Apros model that is performed always when the simulation is started if new modules are added to the Apros model or properties of existing ones are modified. The initialisation includes importing necessary Python modules, loading Keras model into the Python environment and defining parameters for scaling the input and output variables of the machine learning model during simulation. On the other hand, the prediction statement is executed at each time step during the simulation to produce predictions of the next output values. In the prediction phase, the scaling parameters are first used to scale the input values to correct scale and then the Keras model is called with the scaled inputs. The predicted output of the data-driven Keras model is then scaled back to original range and assigned to SCL variables that are used to transfer the output values to corresponding Apros modules.

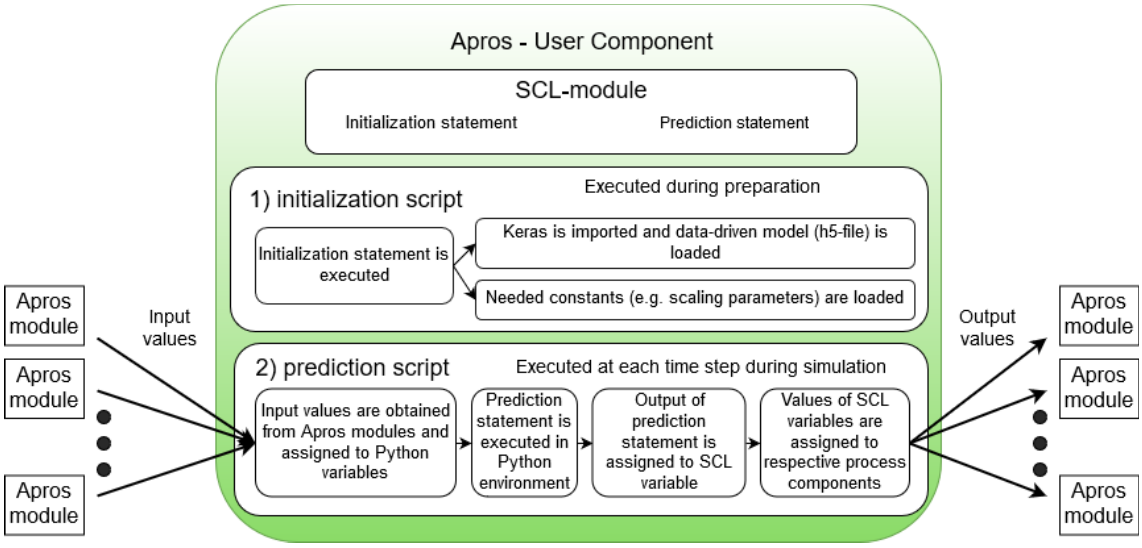


Figure 21. Initialisation and prediction SCL-scripts that are needed to run the machine learning model in Apros using the Python binding.

5.2 Case 1 – Water tank with liquid level control

The first case study considers a water tank model in which there is one continuous flow into the tank and one continuous flow out of the tank as shown in Figure 22. Height of the water tank is 10 m and its cross-sectional area is 0.8 m². The inlet pipe is 6 m long with a control valve in the beginning. The liquid level is measured and controlled with a PI-controller that gives control signal to the control valve. Liquid level setpoint values are limited to range between 4 and 6 m. A pump is used in the outlet pipeline to force flow out from the tank. After the pump, there is a control valve for controlling the outlet flow with a PI-controller, which setpoint in this study is fixed to 45 kg/s. A check valve after the outlet control valve prevents the flow from turning back towards the tank. Both PI-controllers were manually tuned for stable operation. Input and output variable selection was made as shown in Figure 22 and the variables are also listed in Table 3

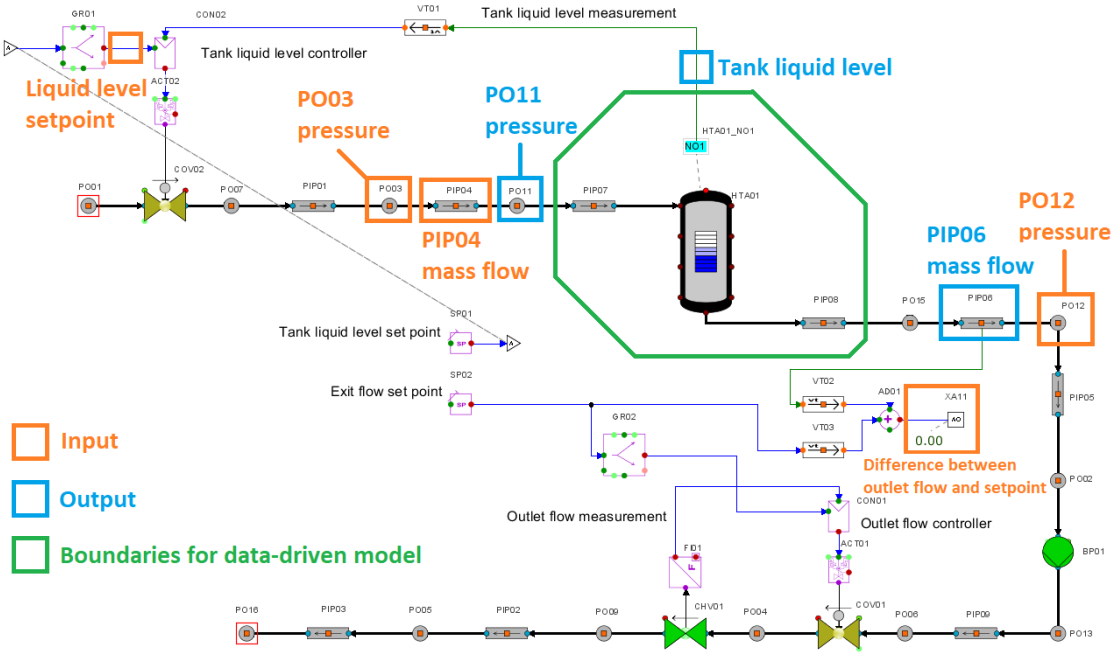


Figure 22. The water tank model studied in case 1. The inputs and outputs are marked in orange and blue respectively. Components inside the green lines are replaced with a machine learning model in this case.

Table 3. Input and output variable selection for data-driven modelling in the 1st case study.

Input variables

#	Apros module	Apros module property	Description	Unit
1	PO03	PO11_PRESSURE	Pressure before the tank (I)	[MPa]
2	PIP04	PI12_MIX_MASS_FLOW	Mass flow before the tank	[kg/s]
3	XA07	ANALOG_VALUE	Liquid level setpoint value	[m]
4	PO12	PO11_PRESSURE	Pressure after the tank	[MPa]
5	XA11	ANALOG_VALUE	Difference between flow after the tank and its setpoint	[kg/s]

Output variables

#	Apros module	Apros module property	Description	Unit
1	PO11	PO11_PRESSURE	Pressure before the tank (O)	[MPa]
2	PIP06	PI12_MIX_MASS_FLOW	Mass flow after the tank	[kg/s]
3	HTA01	TA13_LIQ_LEVEL	Liquid level in the tank	[m]

As the tank outlet flow controller setpoint is fixed during the experiments, the only excitation variable for the system is the tank liquid level setpoint. Thus, the experiment design for case study 1 was a simple tailor-made design that included totally 23 experiments. The liquid level setpoint of the tank was changed within specified low and high limits, 4 and 6 m, respectively (see Figure 23). The training and validation datasets were generated in a single experiment run. Data from the first 17 experiments were used in training and the last 6 experiments in validation of the models. After each change in the setpoint value, the Apros model was simulated for 70 min which was found out to be enough for the flows and liquid level to reach steady-state. Exception to this is the first experiment that was simulated for 2 min, although the first experiment in the validation dataset include almost 70 min of simulation. Sampling time was set to 0.2 s for the data logging and therefore the training and validation datasets consists of 336 600 and 126 000 time steps respectively. This counts to total amount of 2 692 800 and 1 008 000 data points with 8 variables respectively for training and validation datasets. Also the setpoints used in generating testing and extrapolation datasets are shown in Figure 23.

Two testing datasets were generated using the same setpoint values. In the first testing dataset, the model was simulated for 70 min after each setpoint change. In the second testing dataset, the simulation time after each setpoint change was shorter (see Figure 31 in Subsection 6.1.4) so that the process did not have time to settle to new steady-state conditions before the next experiment was started. The simulation time varied between 6.66 and 60 min. This dataset was generated to test the generalisation ability of the models. The generation of extrapolation dataset included experiments in which the liquid

level setpoint exceeded the limits (4–6 m) that were used in the generation of training dataset. Setpoint values in testing and extrapolation experiments are shown in Figure 23. The testing datasets consist of 33 720 and 9720 time steps respectively and the extrapolation dataset include 33 720 time steps. Thus, the total amounts of data points in these datasets are 1 348 800, 388 800 and 1 348 800 respectively. It should be noted that although the experiment design contains step changes in the setpoint value, the actual values that are sent to the respective controller are fed through a gradient module in Apros to smooth out the controller behaviour. The gradient module allowed the liquid level setpoint to change 0.25 m/min, i.e. a setpoint change from 5 to 6 m took 4 min.

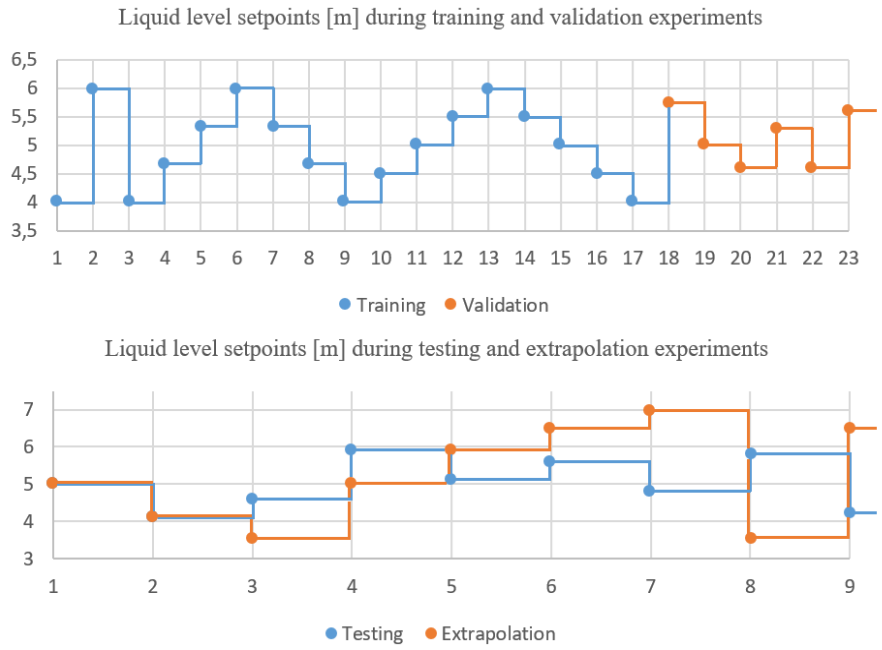


Figure 23. Experiment design used in the data generation for data-driven modelling in case study 1, showing the liquid level controller setpoint values that were used in generation of the datasets.

In the first case study, linear ARX and NARX type of neural network models were developed utilising random search in hyperparameter optimisation (see Table 4). A total of 500 randomly generated hyperparameter combinations were used to build linear ARX models. Development of the NARX model was executed in two steps. In the first step, 1400 models were built (*NARX first* in Table 4) and then based on the results, the second search was executed in a bit different space. In the second search, 200 models were built.

Adam optimiser that is included in Keras was used in the weight optimisation during model building. The used linear ARX model structure did not include a hidden layer but the input neurons are directly connected to the output neurons that include a linear activation function. This means that the outputs of the network are linear combinations of the inputs with each having their own weights that are optimised during the network training. NARX models include one hidden layer in which a hyperbolic tangent function was used as the activation function. For training of both model types, the input samples were used in shuffled order that might increase the efficiency of the training according to (Goodfellow et al. 2016). Batch size was set to 8 800 samples which is the amount of input samples for which the predictions are calculated before updating the weights during a training epoch. Thus, a training epoch includes 52 batches of size 8 800 and the rest of samples are not used in the training. The batch size affects the neural network training time. Therefore, the batch size was increased in initial hyperparameter optimisations until the training time did not decrease anymore. The batch size was not noticed to affect the network performance in the case studies. Weights of the network were initialised by drawing values randomly either from uniform or normal distribution. The mean and standard deviation of the normal distribution are 0 and 0.05 respectively whilst the uniform distribution is characterized by minimum value of -0.05 and maximum value of 0.05. (Chollet 2015) The results of the first case study are presented in Chapter 6.

Table 4. Hyperparameter spaces used in hyperparameter optimisation in the first case study to find the best ARX and NARX neural network models.

Hyperparameter	**ARX**	**NARX first**	**NARX final**
N hyperparameter comb.	500	1400	200
N input lags	1	1–2	1
N of output lags	1	1–2	1
N of hidden layers	No hidden layer	1	1
N of hidden neurons	None	3–6	6–8
Weight init.	randUn, randNo	randUn, randNo	randUn, randNo
Learning rate	$1e^{-4}$–$5e^{-3}$	$1e^{-4}$–$1e^{-3}$	$4e^{-4}$–$1e^{-3}$
Learning rate decay	0	$1e^{-8}$–$1e^{-7}$	0
N of training epochs	10–150	250–1000	100–1000

5.3 Case 2 – Methanation reactor in a power-to-gas process

The second case study in this research is a power-to-gas (P2G or PtG) process. Extensive review of electrolysis and methanation technologies that can be utilised in a power-to-gas process can be found in (Götz et al. 2016). Power-to-gas process is designed to produce hydrogen (H_2) or methane (CH_4) using electricity. The process is two-staged. In the first stage, hydrogen and oxygen are produced in water electrolysis and in the second stage, the produced hydrogen is converted to methane in methanation process that utilise carbon monoxide (CO) or carbon dioxide (CO_2) as a second reactant. Alkaline electrolysis (AEL), polymer electrolyte membranes (PEM) and solid oxide electrolysis (SOEC) are technologies used in water electrolysis in P2G process. Hydrogen produced in the electrolysis process can also be stored before feeding it to the methanation process. This makes power-to-gas process suitable to be used as a seasonal energy storage as surplus electricity can be converted into gas that is more convenient to store for later use than electricity. (Götz et al. 2016)

Biological or catalytic reactor can be used in the methanation process. Quality requirement for the methanation product, i.e. substitute natural gas (SNG), can be compared to that of natural gas that is distributed into a gas grid. Methane concentration is usually over 80 % in natural gas which also contains other hydrocarbons such as ethane, propane and butane that increase its calorific value and inert CO_2 or N_2 that lowers the calorific value. (Götz et al. 2016)

An Apros model of a two-stage power-to-gas process is studied in the second case study. In the Apros model (see Figure 24 for an overview), hydrogen is produced in PEM electrolysis units and after that the hydrogen is compressed before feeding it into the storage tanks. The storage unit consists of low pressure (3 MPa), medium pressure (12 MPa) and high pressure (30 MPa) tanks with 6, 30 and 10 m^3 volumes respectively.

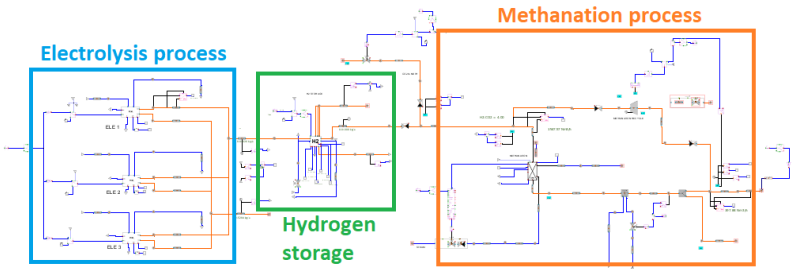


Figure 24. Overview of the PtG Apros model. Electrolysis unit operation process is marked in blue, hydrogen storage in green and methanation process in orange.

The methanation unit operation model that is under the scope in this study, is presented in Figure 25. The main part of the model is the methanation reactor that is a custom made Apros User Component. In the feed pipeline, pure H_2 and CO_2 are first mixed together and then recycled CH_4 is added to the mix. Hydrogen feed to the methanation process is controlled with a PI-controller. The molar ratio of H_2 and CO_2 is controlled with a PI-controller by adjusting the CO_2 flow. The mixture of H_2, CO_2 and CH_4 is fed into the methanation reactor whose temperature is controlled by using cooling steam. Mass flow of the cooling steam is controlled with a PI-controller.

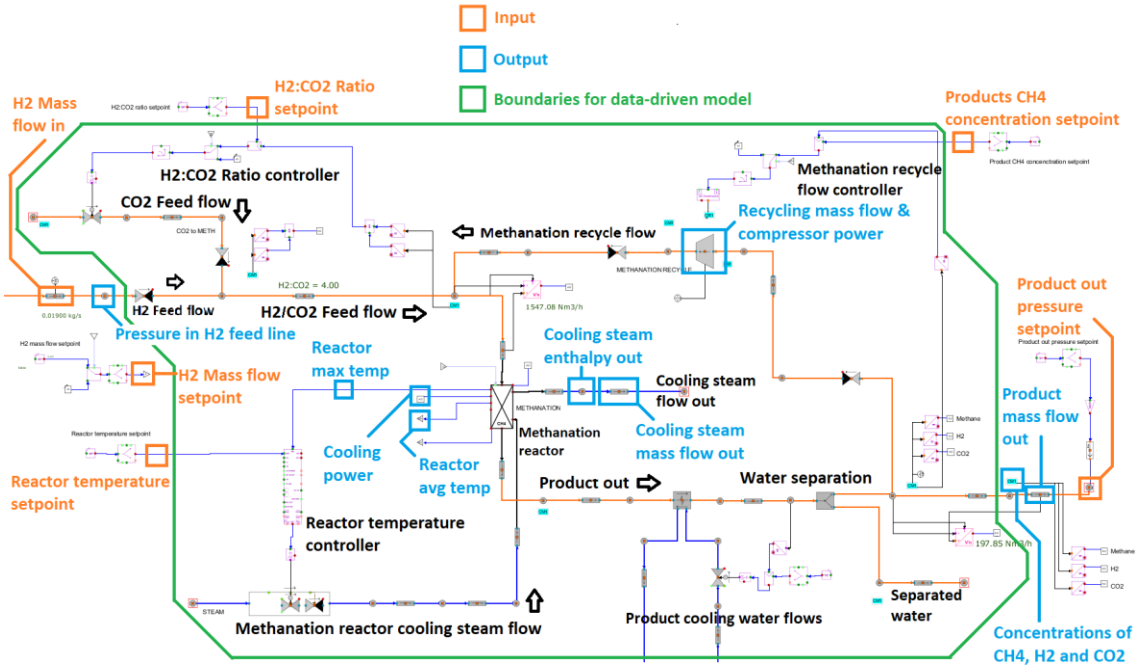


Figure 25. Apros diagram of the studied methanation unit operation process that is part of the power-to-gas Apros model, showing the input and output selection.

Product from the reactor is cooled down to about 25 °C in a separate heat exchanger for water separation. The temperature of the product is controlled with a PI-controller that adjusts mass flow of the cooling water. After the water has been separated from the product, part of it is recycled back to the reactor feed flow. Rest of the product is fed out of the process through a pipe where the H_2, CO_2 and CH_4 concentrations are measured. The recycling flow is controlled by the CH_4 in the product for which a setpoint is given.

Input and output variable selection in the second case study was made as shown in Figure 25 and the variables are also listed in Table 5 and Table 6. The inputs include hydrogen mass flow in to the unit operation process, four setpoints for the controllers that were mentioned in the previous section and the pressure of the product which is not calculated during the simulation but changed manually to modify the conditions in the process.

Table 5. Input variables selected in the second case study.

#	Apros module	Module property	Description	Unit
1	PIP03	PI12_MIX_MASS_FLOW	H2 feed mass flow measured	[kg/s]
2	XA271	ANALOG_VALUE	H2 feed mass flow setpoint	[kg/s]
3	XA257	ANALOG_VALUE	Methanation reactor maximum temperature	[°C]
4	XA258	ANALOG_VALUE	H2:CO2 molar ratio setpoint	[1]
5	XA260	ANALOG_VALUE	Product CH4-% setpoint	[1]
6	PO276	PO11_PRESSURE	Product pressure	[MPa]

Table 6. Output variables selected in the second case study.

#	Apros module	Module property	Description	Unit
1	PO11	PO11_PRESSURE	H2 feed in pressure	[MPa]
2	PIP06	PI12_MIX_MASS_FLOW	Product mass flow out	[kg/s]
3	PO18_CM1	FGCO_CONCENTRATION(2)	Product CH4-%	[1]
4	PO18_CM1	FGCO_CONCENTRATION(6)	Product H2-%	[1]
5	PO18_CM1	FGCO_CONCENTRATION(4)	Product CO2-%	[1]
6	XA289	ANALOG_VALUE	Reactor average temperature	[°C]
7	XA180	ANALOG_VALUE	Reactor maximum temperature	[°C]
8	PIP109	PI12_MIX_MASS_FLOW	Cooling steam mass flow out	[kg/s]
9	PO14	PO11_STEAM_ENTH	Cooling steam enthalpy out	[kJ/kg]
10	XA285	ANALOG_VALUE	Reactor cooling power	[kW]
11	CR03	CM11_MIX_MASS_FLOW	Recycling mass flow	[kg/s]
12	CR03	CM11_POWER	Recycling compressor power	[MW]

To cause excitation in the system and reveal the dynamics of the process, the values of inputs 2–6 were changed during data generation that was executed according to inscribed type of Central Composite design of experiments presented in Subsection 3.1.1. The

experiment design includes 46 different operating conditions, i.e. setpoint value combinations (see Figure 26). The values of the excitation variables (input variables 2–6) are varied during the experiments within the limits shown in Table 7. First datasets were generated using wider limits but it led to unstable operating conditions. Thus, the training data generation process was executed in an iterative manner to test what are the low and high limits for the input factor to keep the Apros process in stable operation.

Table 7. Low and high limits and gradient values for the excitation variables in the second case study that were used in the training dataset generation.

Description	Low limit	High limit	Unit	Gradient [unit/min]
H2 feed mass flow setpoint	0.02	0.03	[kg/s]	$5e^{-3}$
Methanation reactor maximum temperature	550	600	[°C]	25
H2:CO2 ratio setpoint	4	4.5	[1]	0.5
Product CH4-% setpoint	93	96.6	[%]	2
Product pressure	0.6	0.6	[MPa]	0.05

It should be noted that although the input variable 5, i.e. the methane concentration in the product, does not have unit in Table 5, the corresponding variable in the experiment design and in Table 7 is reported in percent. As in the first case study, also in this case the setpoint values are fed to the controller through a gradient module in Apros for smoother operation. The gradient values are included in Table 7. For example the setpoint of methanation reactor maximum temperature controller is allowed to change 25 °C/min.

In each experiment and after the changes in the setpoint values, the Apros model was simulated for 200 min which was found out to be enough for the outputs to settle in new steady-state operation. In Figure 26 and 27, the first experiment marked by “1”, is the initial condition in which the process is simulated for 200 min. After that, the first transient occurs, i.e. the H_2 feed setpoint is changed. Sampling time was set to 0.2 s for the data logging which leads to 2 760 000 time steps, which results to a total amount of 49 680 000 data points with 18 variables. However, the training, validation and testing datasets were each sampled down from 0.2 to 5 s sampling time which reduced the total amount of data points to 1 987 200. Experiment designs for generating validation and testing datasets were tailor made and contained 11 experiments both (see Figure 27), resulting in 26 400 time steps with total amount of 475 200 data points after down sampling. The experiment designs are shown in Figure 27 which shows the levels for each input factor, i.e. excitation variables, in the experiments.

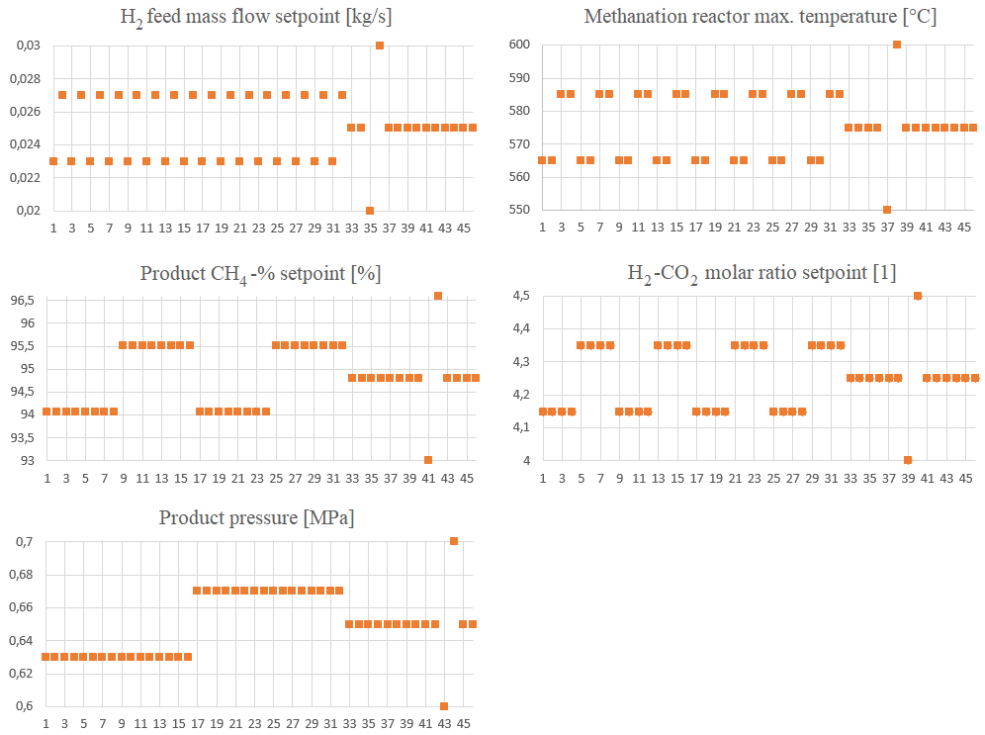


Figure 26. CCI experiment design to generate the training dataset, illustrated for each five input factors, showing the five levels of each.

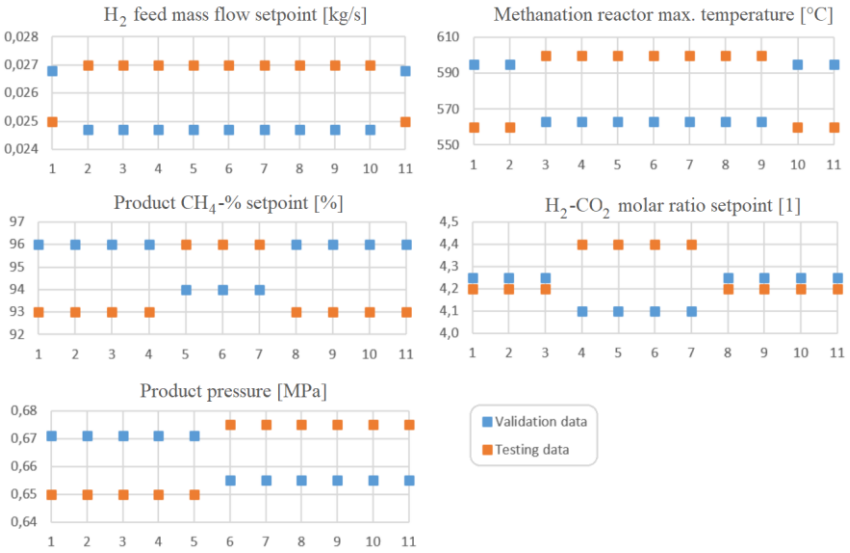


Figure 27. Setpoint levels of the five input factors in the experiment designs for validation and testing data, showing each factor having two levels.

The model building was executed in multiple hyperparameter optimisation runs, utilising random search having different search space in each run. The hyperparameter search

spaces for each model type presented in Table 8 are the ones that were used in deriving the final versions of each model type. The results that were obtained using the hyperparameters are presented in Section 6.2. Other hyperparameter optimisation runs and their search spaces together with the best single model obtained with each run are shown in Appendix 1. The selection of the final models was made by selecting the one with the lowest average NRMSE value which had input lag and one output lag values only from the previous time step included in the inputs.

Table 8. Hyperparameter optimisation runs of which the final models of each model type were selected. Other optimisation runs are in the Appendix 1.

Hyperparameter	**ARX-2**	**NARX-6**	**LSTM-3**	**GRU-3**
N hyperparameter combs	2119	2213	527	267
N of hidden neurons	-	64–128	64–128	64–128
Learning rate	$1e^{-4}$–$5e^{-3}$	$2e^{-4}$–$1e^{-3}$	$2e^{-4}$–$1e^{-3}$	$2e^{-4}$–$1e^{-3}$
N of training epochs	10–150	100–250	200–500	200–500
N_dropout	-	1	1–3	1
Dropout rate	-	0.05–0.2	0.05–0.3	0.05–0.2
Maxnorm	-	0–5	0–5	0–5

6 RESULTS

This study includes a literature review on matters related to data-driven modelling and especially modelling with neural networks. In the case studies, suitability of selected neural network architectures on multi-step-ahead predicting of dynamic behaviour of two processes. The case specific results are divided in sections which present the performance of the models on training, validation and testing datasets. The results of the two case studies are presented in Sections 6.1 and 6.2. In this section, generic findings related to both cases are presented.

Early stopping of the neural network training process was tested in some hyperparameter optimisation experiments and it did not provide better results compared to experiments without early stopping. Keras calculates one or more user specified error metrics which can be used in monitoring the neural network training process. The metrics are calculated on the training and validation datasets. The problem when deriving a model with the best multi-step-ahead accuracy is that the metrics Keras calculates are not useful in the same way as when building one-step-ahead models. This is because the one-step-ahead accuracy that Keras calculates do not correlate to multi-step-ahead accuracy. To demonstrate this, one-step-ahead and multi-step-ahead NRMSE values of 800 linear ARX models on the validation dataset of the second case study are plotted in Figure 28. The best models in terms of average NRMSE value in multi-step-ahead configuration are on the left and the worst on the right. It is visible from Figure 28 that the models with the lowest one-step-ahead NRMSE are not the best models in multi-step-ahead configuration. The models with the best multi-step-ahead accuracy, i.e. NRMSE ≈ 1.6 %, have NRMSE of 0.5…0.75 % one-step-ahead configuration. As the multi-step-ahead NRMSE increase when going to the right in Figure 28, it can be seen that the one-step-ahead NRMSE have a decreasing trend as shown by the black line. In addition to that, the variance of one-step-ahead NRMSE seems to increase as the multi-step-ahead NRMSE increase. This is expected because in the one-step-ahead configuration, the ground truth values of outputs are used as the autoregressive inputs and in the multi-step-ahead configuration, the autoregressive inputs are network output values from previous time steps.

Due to this, a test bench for testing the models in multi-step-ahead configuration was implemented. In the test bench, the models are used as simulators, i.e. their outputs are fed back to their inputs as the models are autoregressive.

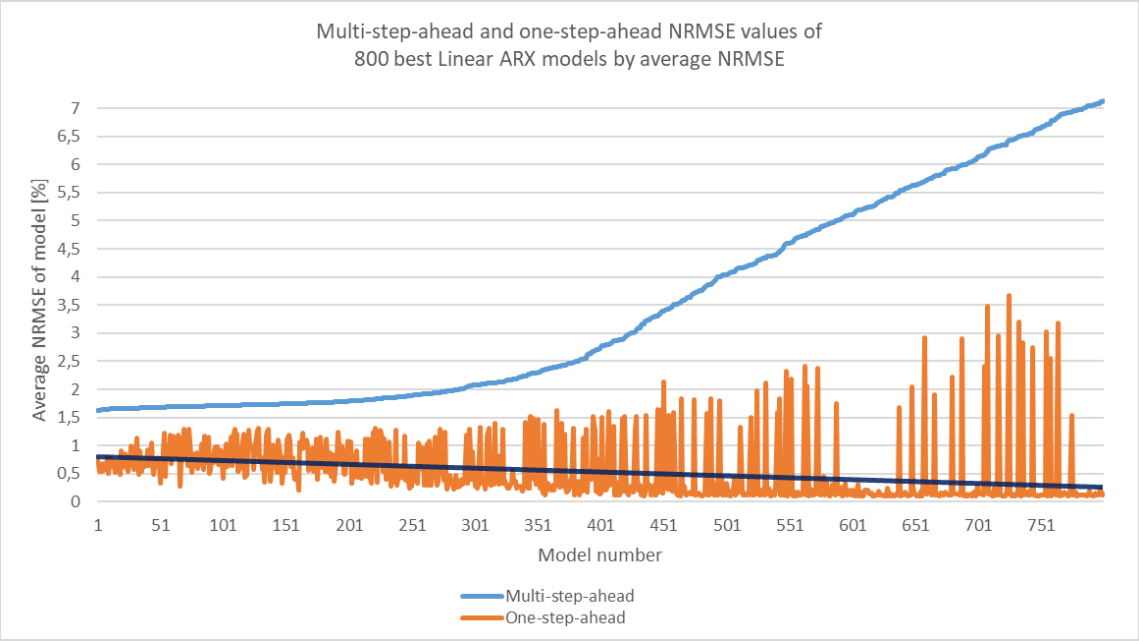


Figure 28. Multi-step-ahead and one-step-ahead NRMSE values for 800 linear ARX neural network models on validation dataset in the second case study, showing that the models with the lowest one-step-ahead errors do not seem to be the best models in multi-step-ahead configuration.

6.1 Case 1 – Water tank with liquid level control

Performance of linear ARX and nonlinear NARX neural network models was studied with a water tank process which includes two control loops – liquid level control that adjusts the inlet flow going to the tank and outlet flow control. The hyperparameter search spaces that were used to obtain these results were presented in Table 4 in Section 5.2 and the hyperparameters of the selected models are described below in Subsection 6.1.1. During hyperparameter optimisation, multi-step-ahead accuracy of each model was measured on the validation dataset and the model with the lowest NRMSE value was selected for testing. Results on validation, multiple testing and training datasets are presented in the following Subsections 6.1.2–6.1.6.

6.1.1 Selected models

The final linear ARX model in the first case study includes 5 exogenous inputs, i.e. input values from the previous time steps, and 3 autoregressive inputs, i.e. output values from the previous time steps. The weights were initialised by drawing randomly values from uniform distribution. During the training, the weights were optimised using Adam training algorithm with learning rate and learning rate decay were 4.061e-3 and 0 respectively. Adam algorithm's β_1 and β_2 parameters were kept at their default values in Keras ($\beta_1 = 0.9$, $\beta_2 = 0.999$). The model was trained for 150 epochs.

The final NARX model included the same inputs as the ARX model but the structure has a hidden layer with hyperbolic tangent activation function to model nonlinearities. The hidden layer in the final NARX model contains 8 hidden neurons and the weights of the network were initialised by drawing randomly values from normal distribution. As with the linear ARX model, Adam was also used with the NARX model to optimise the weights during training. The learning rate of Adam was 7.901e-4 and other parameters as with the linear ARX model. The NARX model was trained for 829 epochs and no regularisation was applied.

6.1.2 Results on validation dataset

The final models which structure and hyperparameters were presented in the previous section, were selected based on their multi-step-ahead performance on the validation dataset. The results of simulating the validation dataset are shown in Figure 29. The average NRMSE value of both linear ARX and NARX models are shown in the title in Figure 29 and the three output specific NRMSE values in their respective subfigures. As the errors are so low, it is hard to see any difference but the NRMSE values show that pressure before the tank has the highest and mass flow after the tank has the lowest NRMSE with both model types. The linear ARX model outperforms the nonlinear NARX model in prediction of each individual output variable. All in all, the accuracy of both models is very good as the average NRMSE values are 0.07 % for linear ARX and 0.11 % for NARX model.

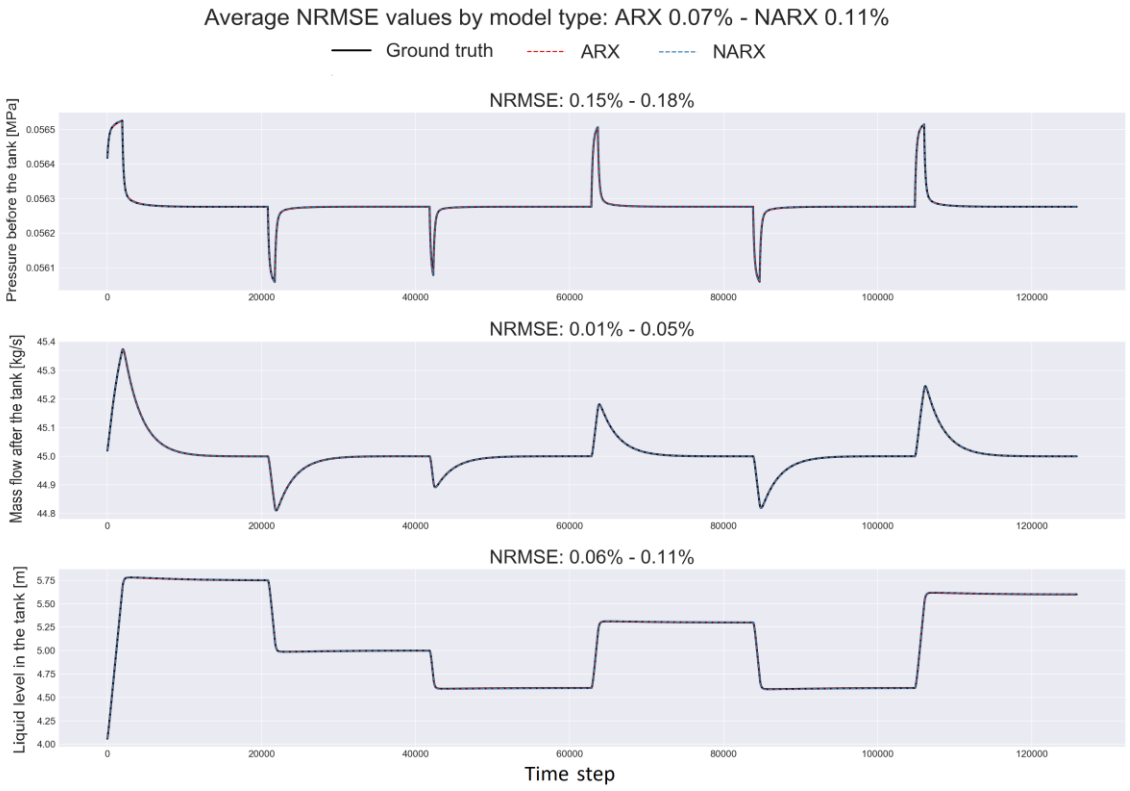


Figure 29. Multi-step-ahead prediction performance of the final ARX and NARX models on the validation dataset, showing slightly lower average NRMSE for the linear ARX.

6.1.3 Results on testing dataset

The next step was to measure the performance of the selected models using an individual testing dataset that was not been used in the hyperparameter optimisation and thus can provide more reliable estimate of the model generalisation ability. The results on the testing dataset are presented in Figure 29 which shows that the linear ARX model achieved the same NRMSE values as it did on the validation dataset. Figure 29 also shows that the NARX model performs even better on the testing dataset than on the validation dataset but still had slightly worse average and output specific NRMSE values than the linear ARX model. The results on the testing dataset suggest that the generalisation ability of both models is sufficient.

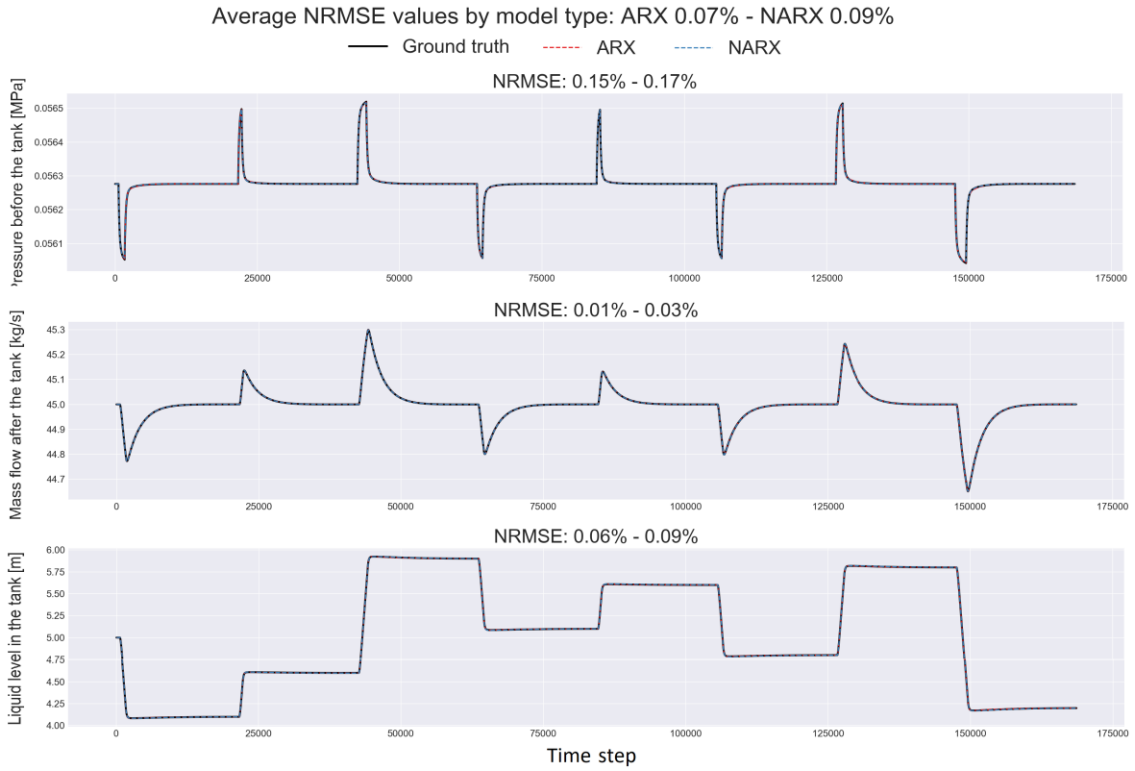


Figure 30. Multi-step-ahead prediction performance of the final ARX and NARX models on the testing dataset.

6.1.4 Results on testing dataset with faster changes

The final models were also tested on a dataset which contained operating conditions that change before the steady state is reached, i.e. the setpoint of the liquid level controller was changed faster than in the validation and testing datasets. The results obtained using this dataset are presented in Figure 31, showing that the final models are able to handle also this kind of situations fairly well. However, the average NRMSE values of both models show deterioration compared to the validation and testing datasets which is expected as the training dataset does not contain any examples on this kind of situations. The only output variable which prediction accuracy is not deteriorated is the mass flow after the tank predicted with the linear ARX model.

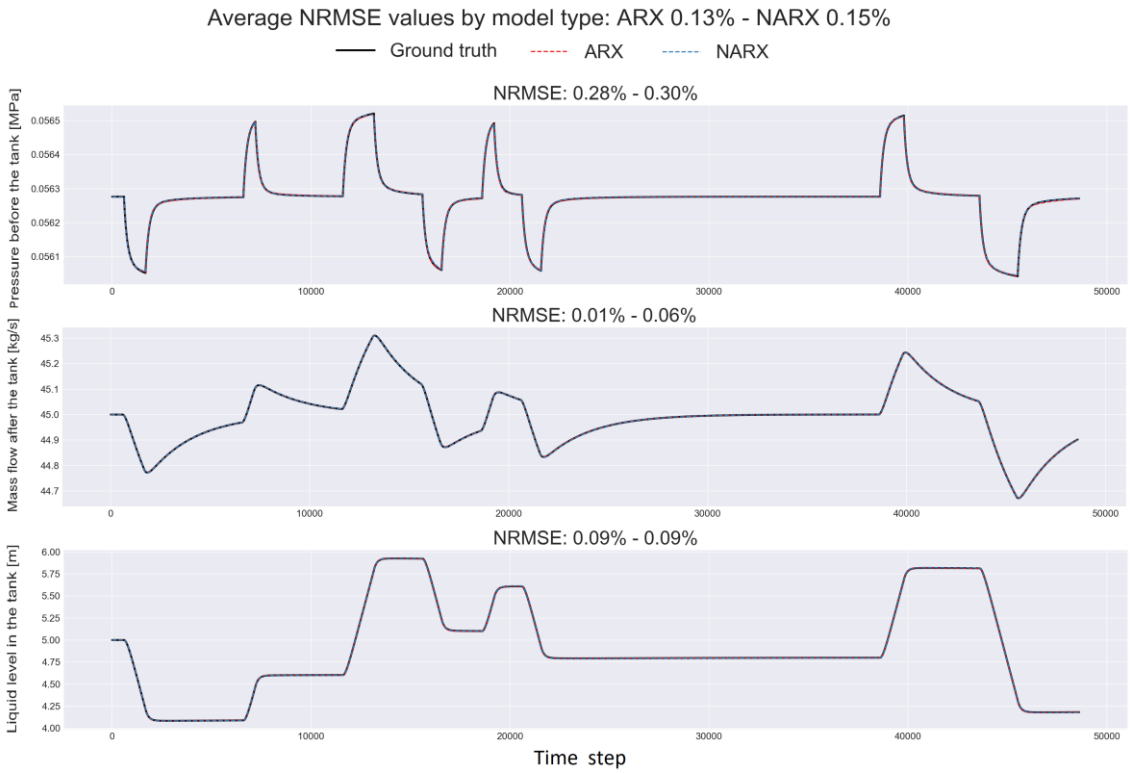


Figure 31. Multi-step-ahead performance of case study 1 models on dataset that contain faster changes in the liquid level setpoint value.

6.1.5 Results on testing dataset with extrapolation

Extrapolation capability of the model was also tested. In the extrapolation dataset, the minimum and maximum setpoint values of the liquid level controller were slightly over the limits (3–7 m) of the other datasets (4–6 m). Results on the extrapolation dataset are presented in Figure 32. The results show more deterioration, i.e. much worse NRMSE values, than the previous test with faster occurring setpoint changes. The average NRMSE of linear ARX model increase from 0.07 to 0.60 % and from 0.09 to 0.92 % respectively with the NARX model. The largest errors with both model types occur at about time step 130 000, when the liquid level setpoint was just set to 7 m. Both of the models predict the pressure before the tank to be lower than the ground truth is. However, both of the models recover from the extrapolation situation.

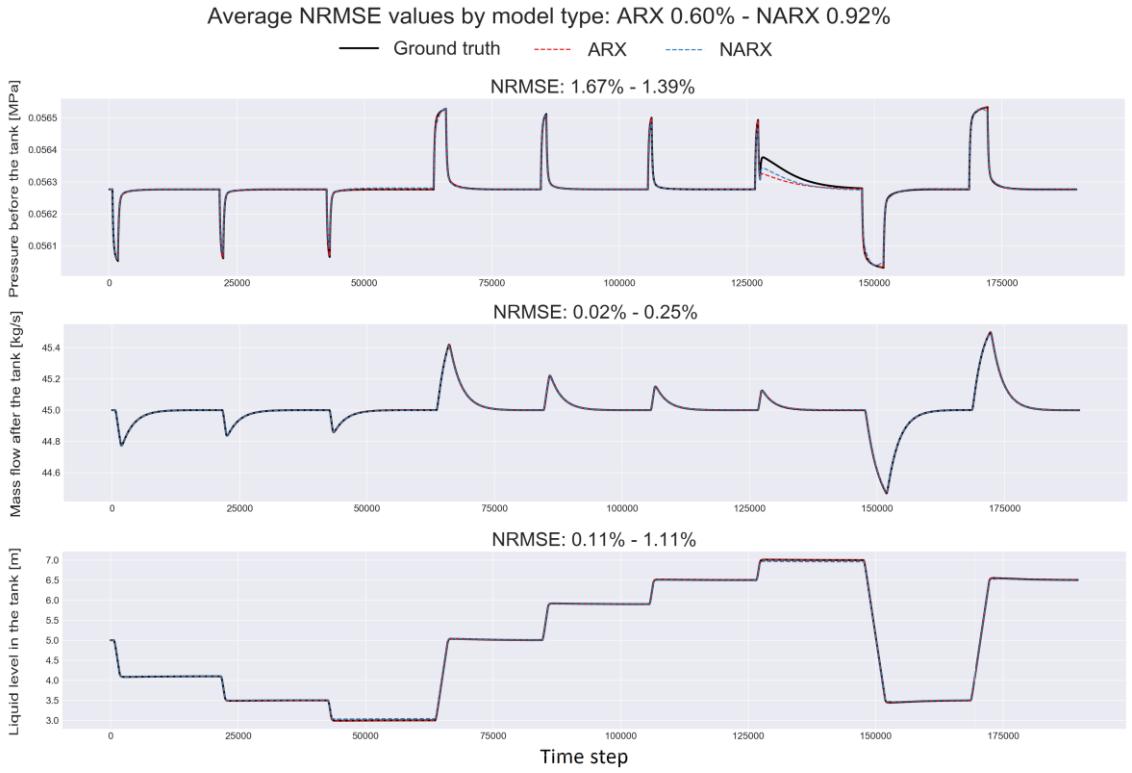


Figure 32. Multi-step-ahead performance of case study 1 model on extrapolation dataset, showing more deterioration on the accuracy of the final model's performance.

6.1.6 Results on training dataset

Accuracy of the models on the training dataset was also measured in multi-step-ahead configuration and the results are shown in Figure 33. The results on the training dataset show slightly higher NRMSE values than on the validation and testing datasets. This is expected because the range of the liquid level setpoint in the training dataset is a bit wider than in the other datasets.

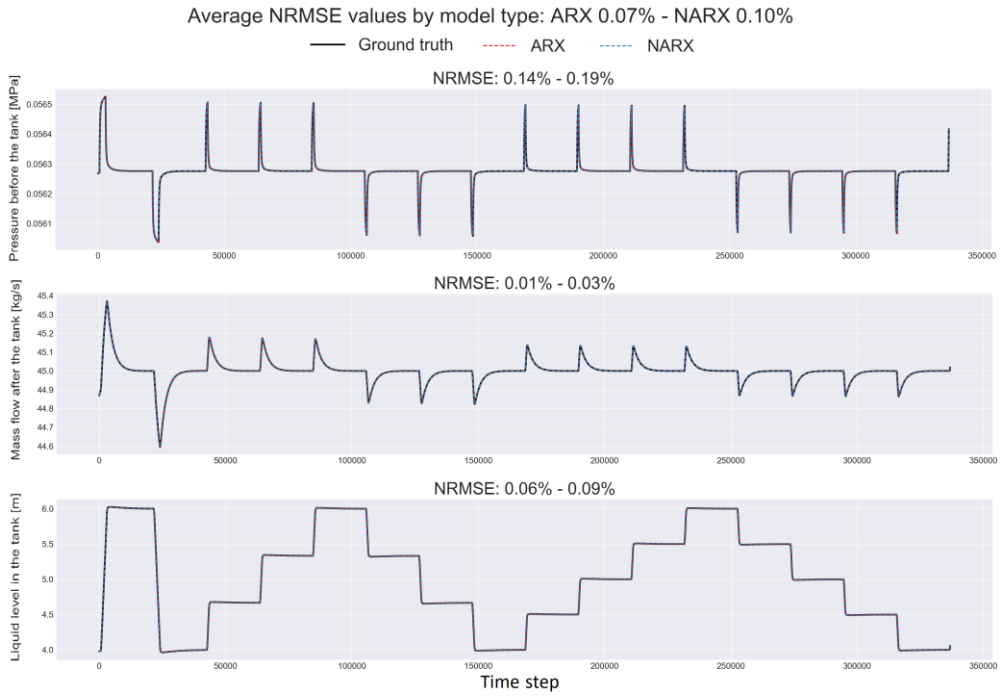


Figure 33. Multi-step-ahead simulation of the final models of case study 1 on the dataset that was used in training the models.

In summary, the performance of both linear ARX and NARX model is very good based on the testing with multiple different datasets. The results are summarized in Table 9 in which *Test. 1* refers to the testing dataset and *Test. 2* to the testing dataset with faster changes. The models were able to extrapolate with only small deterioration in the accuracy and are able to recover from the extrapolation situations back to steady state well. The results suggest that linear ARX model is the best model choice for this case, having better accuracy than the nonlinear NARX model and being much simpler in terms of the number of model parameters.

Table 9. Summary of the NRMSE values of both model types on the five datasets.

Model type	Dataset	NRMSE values of output variables			
		1	2	3	Avg.
ARX	Valid.	0.15	0.01	0.06	0.07
	Test. 1	0.15	0.01	0.06	0.07
	Test. 2	0.28	0.01	0.09	0.13
	Extrap.	1.67	0.02	0.11	0.60
	Train.	0.14	0.01	0.06	0.07
NARX	Valid.	0.18	0.05	0.11	0.11
	Test. 1	0.17	0.03	0.09	0.09
	Test. 2	0.30	0.06	0.09	0.15
	Extrap.	1.39	0.25	1.11	0.92
	Train.	0.19	0.03	0.09	0.10

6.2 Case 2 – Methanation reactor in a power-to-gas process

Linear ARX, NARX, LSTM and GRU multi-step-ahead neural networks were developed in the second case study and the results obtained with the final models are presented in this section. In this case study, multiple hyperparameter optimisation runs were executed. The results of each model type and run are presented in Appendix 1. The results that are presented in this section are produced with models that were selected based on their performance on the validation dataset and are from hyperparameter optimisation runs ARX-2, NARX-6, LSTM-3 and GRU-3, names referring to the ones in Appendix 1. The hyperparameter spaces that were used in these runs were presented in Section 5.3 in Table 8. During the testing of these selected models on the three different datasets, the time to make the predictions was measured. The average time to make one prediction with different model types are presented in Table 10 showing that the linear ARX network is the fastest and LSTM the slowest. This is expected as the ARX model is the simplest one and LSTM the most complex. The models were tested on validation, testing and training datasets and the results are presented in Subsections 6.2.2–6.2.4.

Table 10. Average time to produce one prediction during multi-step-ahead prediction. The results are obtained using the selected models.

	Prediction time [ms]		
Model type	**Training**	**Validation**	**Testing**
ARX	0,23	0,20	0,22
NARX	0,27	0,28	0,29
LSTM	0,43	0,44	0,41
GRU	0,40	0,41	0,39

6.2.1 Selected models

The final linear ARX model in the first case study include 6 exogenous inputs, i.e. input values from the previous time steps, and 12 autoregressive inputs, i.e. output values from the previous time steps. The weights were initialised by drawing randomly values from normal distribution. During the training, the weights were optimised using Adam algorithm with learning rate and learning rate decay were 4.74e-2 and 0 respectively. Adam algorithm's β_1 and β_2 parameters were 0.8942 and 0.9962 respectively. The model

was trained for 36 epochs which provided the best multi-step-ahead accuracy based on the validation set.

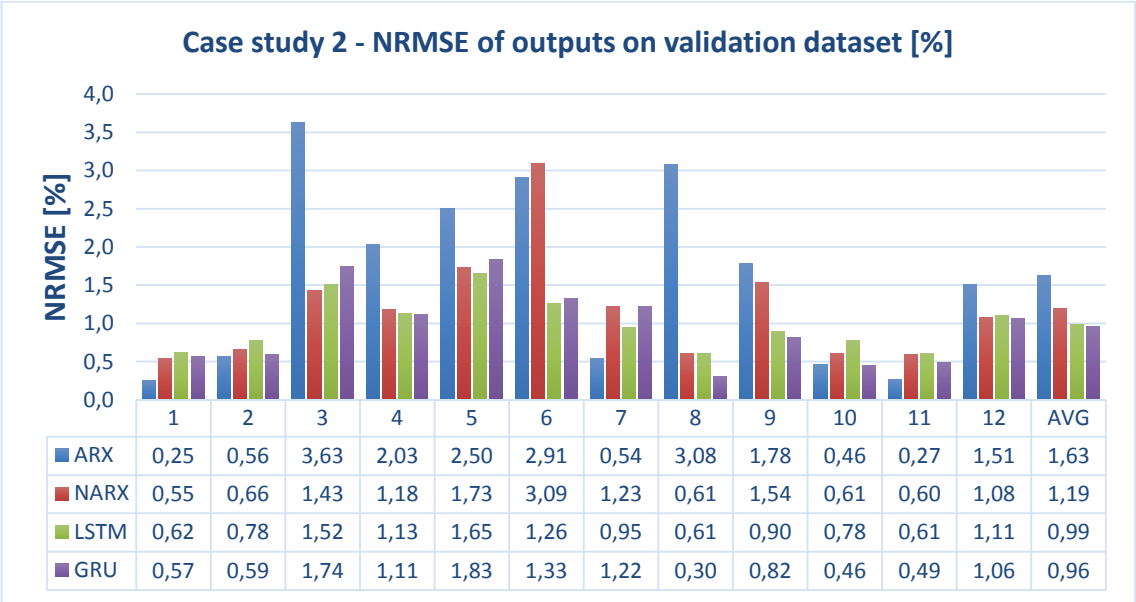
The selected NARX, LSTM and GRU models each have hyperbolic tangent activation function in the hidden layer and the weights of the network are initialised by drawing values randomly from uniform distribution. As with linear ARX model, Adam algorithm was used as weight optimiser with learning rate decay set to 0. Adam's parameters β_1 and β_2 parameters during training of each of the three nonlinear models were set to 0.9 and 0.999 respectively. The rest of the hyperparameters of the final NARX, LSTM and GRU models are presented in Table 11.

Table 11. Hyperparameters of the final NARX, LSTM and GRU models.

Hyperparameter	**NARX**	**LSTM**	**GRU**
N of hidden units	119	86	93
Learning rate	8.245e-4	5.654 e-4	7.105 e-4
N of training epochs	178	241	349
N_dropout	1	1	1
Dropout rate	0.051	0.103	0.065
Maxnorm	4	3	4

6.2.2 Results on validation dataset

Accuracy of the final models presented in the previous section was tested in multi-step-ahead configurations using the validation, testing and training datasets. The results on the validation dataset are presented in Figure 34 and Figure 35 showing that the lowest average NRMSE value was achieved with the GRU model (0.96 %) and the highest with the linear ARX model (1.63 %). The output variables are marked in Figure 34, Figure 36 and Figure 38 using numbers 1–12. The respective variable names can be found in Section 5.3 in Table 6.



	1	2	3	4	5	6	7	8	9	10	11	12	AVG
ARX	0,25	0,56	3,63	2,03	2,50	2,91	0,54	3,08	1,78	0,46	0,27	1,51	1,63
NARX	0,55	0,66	1,43	1,18	1,73	3,09	1,23	0,61	1,54	0,61	0,60	1,08	1,19
LSTM	0,62	0,78	1,52	1,13	1,65	1,26	0,95	0,61	0,90	0,78	0,61	1,11	0,99
GRU	0,57	0,59	1,74	1,11	1,83	1,33	1,22	0,30	0,82	0,46	0,49	1,06	0,96

Figure 34. NRMSE values of final models and individual output variables on the validation dataset for each model type.

As it can be seen in Figure 34 and Figure 35, there are significant differences between model types and their prediction accuracy for individual output variables. For example, the simplest linear ARX model outperforms the nonlinear models in predicting output variable 1 (H_2 feed pressure), 7 (reactor maximum temperature) and 11 (Recycling mass flow) but it still has the worst overall performance. The biggest difference between the linear ARX model and the nonlinear models is in predicting output variable 8 (Cooling steam outlet mass flow). Although having the lowest average NRMSE, the GRU model performs worse in prediction of output variables 3 (CH_4-% of the product), 5 (CO_2-% of the product) and 6 (reactor average temperature) than NARX or LSTM.

The methanation process dynamics is known to contain nonlinearities which supports the results shown in Figure 34 as the nonlinear models predict the fraction of CH_4, H_2 and CO_2 in the product more accurately than the linear model. This can be easily also seen in Figure 35 in which the predictions of the final models are plotted together with the ground truth values. On average, the NARX neural network model seems to perform worse than LSTM and GRU based on the validation dataset results.

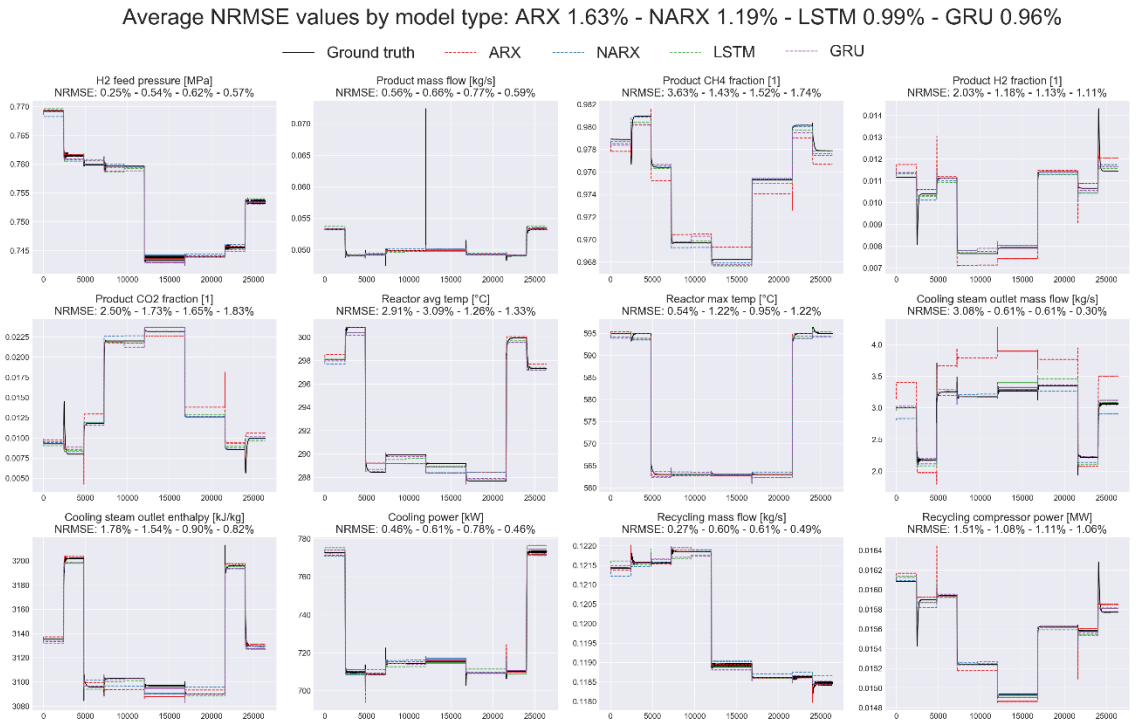
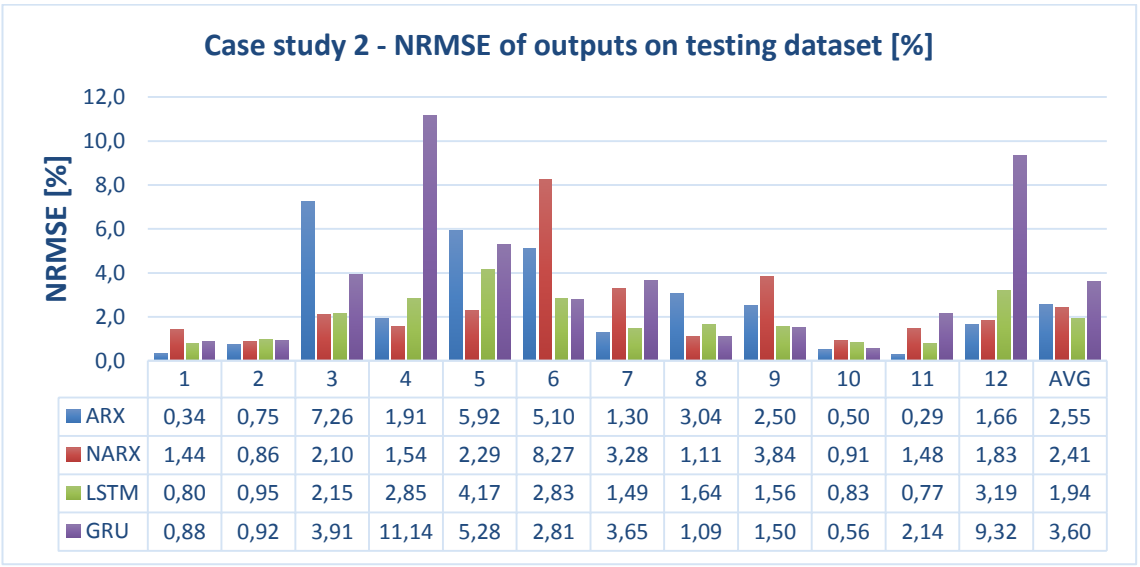


Figure 35. Multi-step-ahead prediction performance of the final models on the validation dataset, with GRU being the most accurate one.

6.2.3 Results on testing dataset

The respective results on the testing dataset are presented in Figure 36 and Figure 37. Based on the results, the model accuracy with each model type is much worse according to the testing dataset compared to the validation dataset. This questions the generalisation ability of the models, although the average NRMSE values are still on quite good level (1.94–3.60 %). The accuracy of GRU network, that had the best accuracy on average on the validation dataset, is the worst model based on the testing dataset. This shows why the independent testing dataset is needed. The difference in the linear ARX model accuracy between the validation and the testing datasets is the smallest compared to other model types.



	1	2	3	4	5	6	7	8	9	10	11	12	AVG
ARX	0,34	0,75	7,26	1,91	5,92	5,10	1,30	3,04	2,50	0,50	0,29	1,66	2,55
NARX	1,44	0,86	2,10	1,54	2,29	8,27	3,28	1,11	3,84	0,91	1,48	1,83	2,41
LSTM	0,80	0,95	2,15	2,85	4,17	2,83	1,49	1,64	1,56	0,83	0,77	3,19	1,94
GRU	0,88	0,92	3,91	11,14	5,28	2,81	3,65	1,09	1,50	0,56	2,14	9,32	3,60

Figure 36. NRMSE values of final models and individual output variables on the testing dataset for each model type.

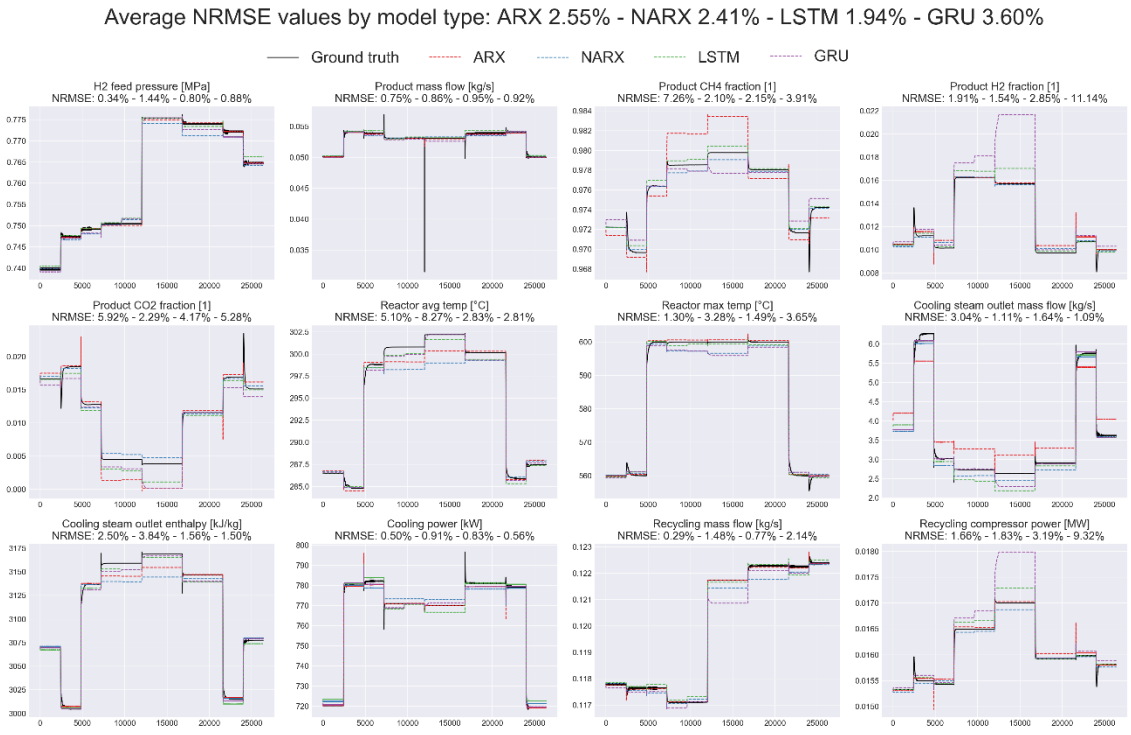
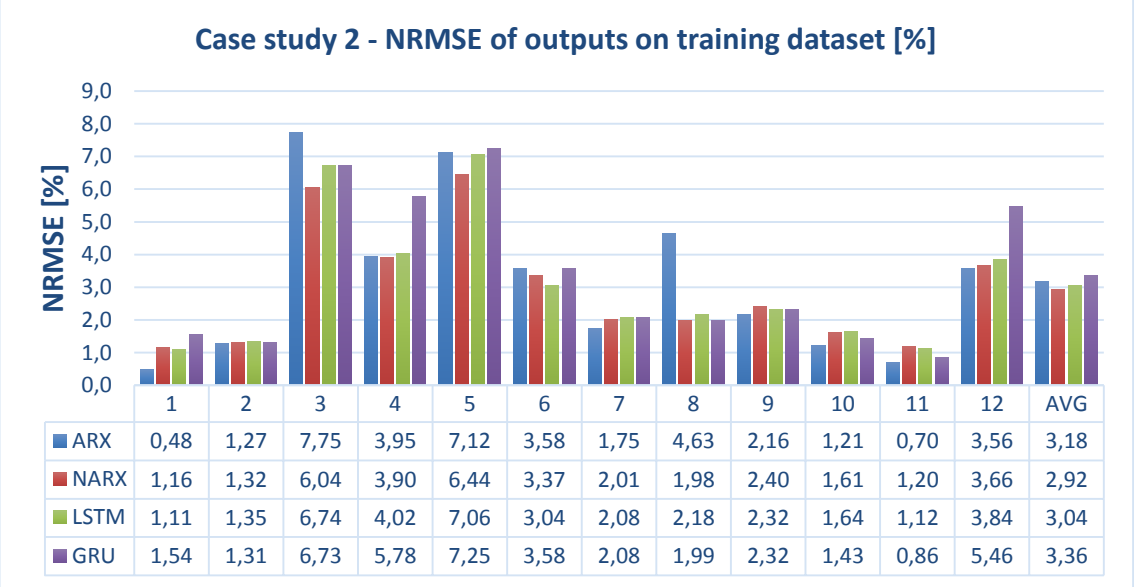


Figure 37. Multi-step-ahead prediction performance of the final models on the testing dataset, showing the LSTM model having the best performance.

6.2.4 Results on training dataset

The accuracy of the final models in the second case study were also tested on the training dataset and the results are shown in Figure 38 and Figure 39. The results are similar to the first case but the effect shows in bigger scale, i.e. the NRMSE value on the training dataset is higher than on the validation or testing datasets. This holds true for each model type but GRU network, which accuracy on the testing dataset is the lowest of the three datasets.



	1	2	3	4	5	6	7	8	9	10	11	12	AVG
ARX	0,48	1,27	7,75	3,95	7,12	3,58	1,75	4,63	2,16	1,21	0,70	3,56	3,18
NARX	1,16	1,32	6,04	3,90	6,44	3,37	2,01	1,98	2,40	1,61	1,20	3,66	2,92
LSTM	1,11	1,35	6,74	4,02	7,06	3,04	2,08	2,18	2,32	1,64	1,12	3,84	3,04
GRU	1,54	1,31	6,73	5,78	7,25	3,58	2,08	1,99	2,32	1,43	0,86	5,46	3,36

Figure 38. NRMSE values of final models and individual output variables on the training dataset for each model type.

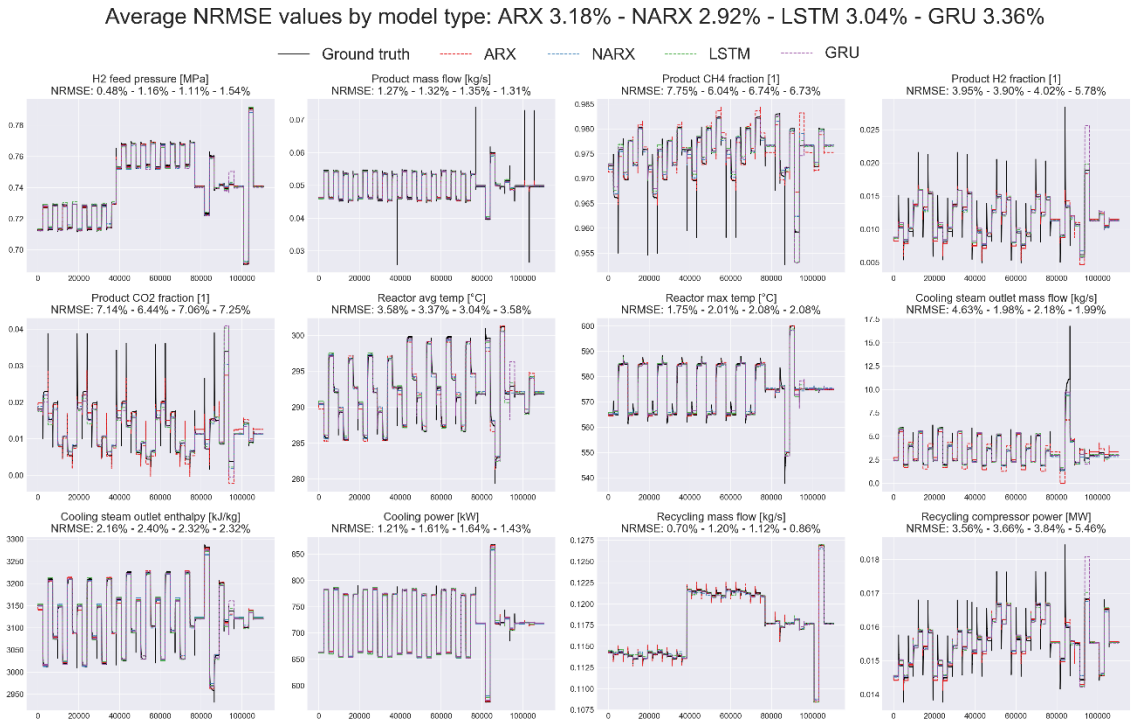


Figure 39. Multi-step-ahead prediction performance of the final models on the training dataset, showing that the NARX neural network has the best performance on that dataset.

In summary, the results of the second case study show more differences between prediction accuracy of individual output variables than the first case study. As expected, some of the output variables are more accurately predicted with nonlinear models and on the other hand some with linear ARX model (see Table 12). The multi-step-ahead accuracy of each model type is worse on the testing and training datasets than on the validation dataset. This is also the case for almost all individual output variables. Overall, the model accuracy achieved with each model type is on a quite good level. The lowest average NRMSE value on the second case study was 3.36 % with GRU model on the training dataset. Although the average values are promising, the prediction accuracy of e.g. output variables 3–6 and 12 are lower on average than others.

Table 12. Summary of the NRMSE values of all four model types on the three datasets.

Dataset	NRMSE values of output variables [%]												
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**Avg.**
	ARX												
Valid.	0.25	0.56	3.63	2.03	2.50	2.91	0.54	3.08	1.78	0.46	0.27	1.51	1.63
Test.	0.34	0.75	7.26	1.91	5.92	5.10	1.30	3.04	2.50	0.50	0.29	1.66	2.55
Train.	0.48	1.27	7.75	3.95	7.12	3.58	1.75	4.63	2.16	1.21	0.70	3.56	3.18
	NARX												
Valid.	0.55	0.66	1.43	1.18	1.73	3.09	1.23	0.61	1.54	0.61	0.60	1.08	1.19
Test.	1.44	0.86	2.10	1.54	2.29	8.27	3.28	1.11	3.84	0.91	1.48	1.83	2.41
Train.	1.16	1.32	6.04	3.90	6.44	3.37	2.01	1.98	2.40	1.61	1.20	3.66	2.92
	LSTM												
Valid.	0.62	0.78	1.52	1.13	1.65	1.26	0.95	0.61	0.90	0.78	0.61	1.11	0.99
Test.	0.80	0.95	2.15	2.85	4.17	2.83	1.49	1.64	1.56	0.83	0.77	3.19	1.94
Train.	1.11	1.35	6.74	4.02	7.06	3.04	2.08	2.18	2.32	1.64	1.12	3.84	3.04
	GRU												
Valid.	0.57	0.59	1.74	1.11	1.83	1.33	1.22	0.30	0.82	0.46	0.49	1.06	0.96
Test.	0.88	0.92	3.91	11.14	5.28	2.81	3.65	1.09	1.50	0.56	2.14	9.32	3.60
Train.	1.54	1.31	6.73	5.78	7.25	3.58	2.08	1.99	2.32	1.43	0.86	5.46	3.36

6.3 Implementation of data-driven model in Apros

In this study, the workflow to implement the final machine learning model in Apros was defined and tested using the final NARX model of the first case study. Despite the very high accuracy of the final model, it started to oscillate quickly when implemented in Apros, getting also the physics-based part to produce bad results. One thing that may cause oscillation is the automatically changing time step in Apros. It should be studied in more detail with a very simple Apros model to find out the reason. Testing also showed that although the Apros Python binding itself is fast, including the Keras model and making predictions using the binding cause the simulation to slow down even though the Keras model was fast in making predictions.

7 DISCUSSION AND FUTURE WORK

The results of the case studies show that high multi-step-ahead accuracy can be achieved with recurrent neural networks in modelling simulated industrial process responses. The results of the first case study show good accuracy with both linear and nonlinear ARX models. In the second case study, the differences between capabilities of different model types are more visible. The behaviour of some output variables can be predicted more accurately with a simple linear ARX model while some others require a nonlinear model. In general, these networks show promising performance in multi-step-ahead predicting.

One thing that may improve the results of this study is using a sophisticated method to select the number of input and output lags even though it would be at the cost of increased number of model parameters. At the moment, the machine learning framework developed in this study allows to choose the number of lags for input and output variables. For example, if the chosen number of input lags is 1, then the input values from previous time step of each input variable is included in the model. Better results could be achieved by allowing the selection for individual variables by e.g. giving the number of lags as a vector containing the number of lags for each variable separately. Including the selection of the number of input and output lags in the hyperparameter optimisation was tested with linear ARX model in the second case study (see hyperparameter optimisation run ARX-3 in Appendix 1), showing promising results in terms of lower NRMSE value compared to the results of linear ARX model shown in Section 6.2. The number of lags for inputs and outputs varied between 1 and 5 in that hyperparameter optimisation and after 540 models were generated, the model with the lowest NRMSE value on validation dataset had NRMSE of 1.41 % which was achieved using 1 input lag and 5 output lags. The best model with 1 input and output lag values achieved NRMSE of 1.63 %.

The second case study showed that the physics-based model and especially the unit operation process that we want to replace with a machine learning model needs to be tested and validated properly before data generation. In this study, the time to create the datasets took reasonable amount of time, e.g. the generation of the training dataset of the second case study took about one day when all the experiments of experiment design were ran sequentially. However, if the Apros models are more complex compared to the ones in the case study, more data for training and testing purposes is required to make

more reliable and accurate models. In addition, the computation time of one experiment may take a longer time if the model is more complex. Therefore, utilising cloud services or other means of decentralized computation to parallelise the data generation would be beneficial. In the same way, the process of model building through hyperparameter optimisation would benefit from parallelization especially if random hyperparameter search is used.

In the second case study, an experiment design that includes only the statistically most significant level combinations of the input factors was used. Using Apros to produce more data of the second case study process by using e.g. full factorial design with three levels, could enhance the model performance and is seen e.g. in (Goodfellow et al. 2016) to be a better option than trying out different training algorithms when the model has learned the previous training dataset well. As can be seen from the results of the second case study, each of the neural networks except GRU has worse multi-step-ahead NRMSE on the training dataset than on the validation and testing datasets. This is as expected because the validation and testing datasets include less experiments than the training data. In addition, the levels of the input factors in those datasets were well inside the minimum and maximum range of the training data experiments. Using more extensive validation and testing datasets could provide a better estimate of the performance of the generated models. To model the second case study system in more detail, the original dataset with 0.2 s sampling time should be used instead of the data down sampled to 5 s sampling time that was used in the case.

Early stopping, presented in the literature part of the study, can be thought of as one way of regularisation as the training is stopped based on some criterion before the neural network either stops improving in terms of validation loss or if the validation loss starts to increase. In this study, the early stopping mechanism was not found useful as the one-step-ahead accuracy that is measured during the training of the neural networks seems not to correlate with the multi-step-ahead accuracy. Instead of early stopping, the number of epochs was included in the hyperparameter search as one factor.

Neural network pruning is a technique in which insignificant weights of a network are removed if they do not contribute any information, i.e. their value is very close to zero. This is one way to reduce the number of model parameters after the model has been built. (Haykin 2009, p. 176–177) In that sense, pruning could be considered in the further

studies as one tool to make the final models lighter if it can be made without sacrificing the accuracy.

The results show that there are differences between the accuracy of model types and between individual output variables. Therefore, it could be beneficial to use a modular model which was presented in Section 4.3, if the model complexity does not increase too much, i.e. the model would be too slow to be implemented in Apros. Modular models could be even put together by using models that have worse NRMSE value but are good in predicting certain output variables. During the hyperparameter optimisation runs, in which the neural network models were trained, the multi-step-ahead accuracy of all models was tested on the validation dataset and saved in a text file for analysing the results later on. These results include the average NRMSE value of all the models and the NRMSE values of the individual output variables. Analysing these results show that NRMSE values of the individual output variables of the model with the best average NRMSE value are not necessarily the global minimum values for that output variable, i.e. the best model structure depends on the output variable predicted. This means that there are worse models in terms of average NRMSE value having individual output variables with lower NRMSE value than that of the best model by average NRMSE. This indicates that if the computational cost does not increase too much by using separate models for different output variables, it could be beneficial to make a modular model of the aforementioned models to achieve even lower average NRMSE. However, as the machine learning model implementation process in Apros requires more work, the speed requirement is not yet known and should be studied to find out if modular models could be used to increase the prediction accuracy.

Many more neural network architectures and model types exist in addition to the ones studied in this research. For example, convolutional neural networks (CNNs) were not considered in this study but according to Bai et al. (2018), CNNs might be good alternative to the studied architectures. Neural networks were used in this work in a way that they recursively produce multi-step-ahead predictions using one-step-ahead predictions, i.e. to act as a simulator. In addition to neural networks, other model structures which can be derived using machine learning methods exist as well. Examples of such are support vector machines (SVM) (Vapnik, 1999), decision trees and random forests that combine multiple decision trees using bagging (Breiman, 2001). However,

methods based decision tree regression does not generally work as well for nonlinear data as neural networks (Tso and Yau, 2007).

The motivation to study the suitability of machine learning models in dynamic process modelling was to find out if machine learning models could be implemented in physics-based modelling and simulation software so that the two model types would work together. In this study, it was found out that high accuracy can be achieved using neural networks in this application. Related to the future work, the model implementation process in Apros should be studied in more detail due to reasons mentioned in Section 6.3. In addition, the accuracy requirement that Apros sets for the machine learning models needs to be defined.

8 CONCLUSIONS

The purpose of this research was to study the suitability of machine learning methods in the dynamic simulation of industrial processes and to find out how these models can be implemented in dynamic physics-based modelling and simulation software Apros. The main motivation is to find out if computationally heavy Apros models can be made faster by replacing unit operation process models with machine learning models. The study was executed in two phases. First, a literature study was made in order to find out which data-driven modelling techniques are suitable for dynamic modelling. The different steps of data-driven modelling were also studied. Then, an experimental study was made in order to test the modelling methods that were selected based on the literature study. In practice, the simulation accuracy of four recurrent neural network architectures were measured with two case study models. The simulation accuracy of the neural network models was compared to the physics-based simulation results.

The process of creating a surrogate model of a physics-based model starts from selecting the unit operation model that is to be replaced with a machine learning model. Next, the input and output variables for the unit operation model are selected and, according to the process and its characteristics, an experiment design is made to generate data for data-driven modelling. In this study, the importance of finding the limits for the operating conditions of the physics-based unit operation model before data generation to avoid generating useless datasets was found through trial and error. After the data has been generated, it needs to be pre-processed. In this study, the pre-processing included normalising the values of each variable into the same scale. A framework that utilises Keras in building neural network models was built as a part of this study. Random search strategy was used in the neural network hyperparameter optimisation. Each of the trained models were tested in a multi-step-ahead configuration on a validation dataset, and after the hyperparameter optimisation the models were ranked by their average NRMSE values. The best of each model type was selected for further testing with an independent testing dataset which is needed to get a more reliable measure of the generalisation ability of the models which tells how well the model perform on data different from the training data.

Recurrent neural networks were chosen to be experimented in the case studies of this research. A linear ARX model and a nonlinear NARX model were compared in the first case study. The results show that the linear ARX model is better in predicting the dynamic behaviour of the studied water tank model with liquid level control. In the second case study, a linear ARX and a nonlinear NARX, LSTM and GRU neural networks were compared in terms of multi-step-ahead accuracy. The process model in the second case study was a methanation unit operation process, which was one part of a larger power-to-gas process. The results of the second case study show lower overall accuracies with machine learning models compared to the first case. This was expected as the methanation process is much more complex than the tank process. The methanation process also contains nonlinearities and because of this, the accuracy of the linear ARX model was lower than the ones of the nonlinear neural networks. Overall, the accuracy of each model type in the second case study was on a good level. However, using modular models could provide even better results if the computational cost of the model does not increase too much.

The NARX model of the first case study was implemented in Apros. However, it was noticed that the simulation actually slows down when Keras model is used through the Python binding of Apros. The physics-based model and neural network model also started to oscillate soon after starting the simulation in Apros. This may be partly due to automatically changing length of time step in Apros and should be considered in the future work. The next step to find out if machine learning models can be implemented in Apros, requires closer look into the Apros Python binding. In addition, the speed requirement for the machine learning model should be specified in order to know how much the machine learning framework needs development in order to achieve it.

REFERENCES

Abrahart, R. J., See, L. M. and Solomatine D., 2008. Data-Driven Modelling: Concepts, Approaches and Experiences. Practical Hydroinformatics: Computational Intelligence and Technological Developments in Water Applications. Berlin Heidelberg: Springer-Verlag: p. 17–30. ISBN 978-3-540-79880-4.

Aguirre, L. A. and Letellier C., 2009. Modeling Nonlinear Dynamics and Chaos : A Review. Mathematical Problems in Engineering 2009: 35 p. Available from: https://doi.org/10.1155/2009/238960 [22.5.2019].

Alexandridis, A., Patrinos, P., Sarimveis, H. and Tsekouras, G., 2005. A Two-Stage Evolutionary Algorithm for Variable Selection in the Development of RBF Neural Network Models. Chemometrics and Intelligent Laboratory Systems 75 (2): p. 149–62. Available from: https://doi.org/10.1016/j.chemolab.2004.06.004 [22.5.2019].

Anon, 2019a. Apros products and services. Available from: http://www.apros.fi/en/product_information_2/products_services [2.5.2019].

Anon, 2019b. Apros – Dynamic Process Simulation. Available from: https://www.simulationstore.com/apros [2.5.2019].

Anon, 2019c. Keras Documentation – Initializers. Available from: https://keras.io/initializers/ [21.5.2019].

Anon, 2019d. Keras Documentation – Core layers. Available from: https://keras.io/layers/core/ [21.5.2019].

Anon, 2019e. Keras code. Available from: https://github.com/keras-team/keras/blob/master/keras/constraints.py#L22 [21.5.2019].

Anon, 2019f. COCOP project – Coordinating Optimisation of Complex Industrial Processes. Available from: https://www.cocop-spire.eu/content/objectives [21.5.2019].

Anon, 2019g. INTENS project – Integrated Energy Solutions to Smart and Green Shipping. Available from: http://intens.vtt.fi/ [21.5.2019].

Anon. 2019h. MathWorks – Documentation – Linear Correlation. Available from: https://se.mathworks.com/help/matlab/data_analysis/linear-correlation.html [21.5.2019].

Anon, 2018a. Apros training material. VTT Technical Research Centre of Finland.

Anon, 2018b. Apros Tutorial. VTT Technical Research Centre of Finland.

Anon, 2015. Tensorflow. Available from: https://www.tensorflow.org/ [4.5.2019].

Bai, S., Kolter, J. Z. and Koltun, V., 2018. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. Available from: https://arxiv.org/abs/1803.01271 [22.5.2019].

Baumann, K., 2003. Cross-validation as the objective function for variable-selection techniques. Trends in Analytical Chemistry 22 (6): p. 395–406.

Bendat, J. S., 1998. Nonlinear systems techniques and applications. New York, John Wiley & Sons: 474p. ISBN 0-471-16576-X

Bergstra, J., and Bengio Y., 2012. Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research 13: p. 281–305.

Bergstra, J., Bardenet, R., Bengio, Y. and Kégl, B., 2011. Algorithms for Hyper-Parameter Optimization. Proceedings of Neural Information Processing Systems (NIPS) 2011: p. 1–9. Available from: https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf [22.5.2019].

Bian, X., Diwu, P., Zhang, C., Lin, L. Chen, G., Tan, X., Guo, Y. and Cheng, B., 2018. Robust Boosting Neural Networks with Random Weights for Multivariate Calibration of Complex Samples. Analytica Chimica Acta 1009: p. 20–26. Available from: https://doi.org/10.1016/j.aca.2018.01.013 [22.5.2019].

Bishop, C. M., 2006. Pattern Recognition and Machine Learning. New York, Springer-Verlag: 738 p. ISBN: 978-0-387-31073-2.

Bowden, G. J., Dandy, G. C. and Maier, H. R., 2005. Input Determination for Neural Network Models in Water Resources Applications, Part 1 – Background and Methodology. Journal of Hydrology 301 (1): p. 75–92.

Breiman, L., 2001. Random forests. Machine Learning 45 (1): p. 5–32. doi: 10.1007/9781441993267_5.

Chang, F-J., Chen, P-A., Lu, Y-R., Huang, E. and Chang, K-Y., 2014. Real-Time Multi-Step-Ahead Water Level Forecasting by Recurrent Neural Networks for Urban Flood Control. Journal of Hydrology 517: p. 836–846. Available from: http://dx.doi.org/10.1016/j.jhydrol.2014.06.013 [22.5.2019].

Chetouani, Y., 2008. Using ARX and NARX Approaches for Modeling and Prediction of the Process Behavior: Application to a Reactor-Exchanger. Asia-Pacific Journal of Chemical Engineering 3 (6): p. 597–605. Available from: https://doi.org/10.1002/apj.118 [22.5.2019].

Cho, K., Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., 2014. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. Available from: http://arxiv.org/abs/1406.1078 [22.5.2019].

Chollet, F., 2015. Keras. Available from: https://keras.io/ [3.5.2019].

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y., 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. Available from: https://arxiv.org/abs/1412.3555 [22.5.2019].

Ćirović, V., Aleksendrić, D. and Mladenović, D., 2012. Braking Torque Control Using Recurrent Neural Networks. Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering 226 (6): p. 754–766. Available from: https://doi.org/10.1177/0954407011428720 [22.5.2019].

Cui, C. and Fearn, T., 2018. Modern Practical Convolutional Neural Networks for Multivariate Regression: Applications to NIR Calibration. Chemometrics and Intelligent Laboratory Systems 182: p. 9–20. Available from: https://doi.org/10.1016/j.chemolab.2018.07.008 [22.5.2019]

Dreyfus, G., 2005. Neural Networks - Methodology and Applications. Berlin Heidelberg, Springer-Verlag: 497 p. ISBN 978-3-540-28847-3

Galelli, S., Humphrey, G. B., Maier, H. R., Castelletti, A., Dandy, G. C. and Gibbs, M. S., 2014. An Evaluation Framework for Input Variable Selection Algorithms for Environmental Data-Driven Models. Environmental Modelling and Software 62: p. 33–51. Available from: http://dx.doi.org/10.1016/j.envsoft.2014.08.015 [22.5.2019].

Gers, F. A. and Schmidhuber, J., 2001. Simple Recurrent Networks Learn Context-Free and Context-Sensitive Languages by Counting. IEEE Transactions on Neural Networks 12 (6): p. 1333–1340. Available from: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=963769 [22.5.2019].

Gers, F. A. and Schmidhuber, J., 2000a. Recurrent Nets That Time and Count. Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium 3: p. 189–194. Available from: http://ieeexplore.ieee.org/document/861302/ [22.5.2019].

Gers, F. A. and Schmidhuber, J., 2000b. Neural processing of complex continual input streams. Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN 2000) Neural Computing: New Challenges and Perspectives for the New Millennium: p. 557–562. Available from: https://ieeexplore.ieee.org/document/860830 [22.5.2019].

Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep Learning. Available from: http://www.deeplearningbook.org/ [3.5.2019].

Graves, A., 2012. Supervised Sequence Labelling with Recurrent Neural Networks.

Studies in Computational Intelligence 385. Berlin Heidelberg, Springer-Verlag: 141 p. ISBN 978-3-642-24796-5.

Guo, T. and Lin, T., 2017. Multi-Variable LSTM Neural Network for Autoregressive Exogenous Model. Available from: https://arxiv.org/abs/1806.06384 [22.5.2019].

Guyon, I., Gunn, S., Nikravesh, M. and Zadeh, L.A., 2006. Feature Extraction - Foundations and Applications. Studies in Fuzziness and Soft Computing 207. Berlin Heidelberg, Springer-Verlag: 778 p. ISBN-10 3-540-35487-5.

Götz, M., Lefebvre, J., Mörs, F., Koch, A. M., Graf, F., Bajohr, S., Reimert, S. and Kolb, T., 2016. Renewable Power-to-Gas : A Technological and Economic Review. Renewable Energy 85: p. 1371–1390. Available from: https://doi.org/10.1016/j.renene.2015.07.066 [22.5.2019].

Hastie, T., Tibshirani, R. and Friedman, J., 2009. The Elements of Statistical Learning Data Mining, Inference, and Prediction, 2nd Ed. New York, Springer-Verlag: 745 p. ISBN 978-0-387-84858-7.

Haykin, S., 2009. Neural Networks and Learning Machines, 3rd Ed. New Jersey, Pearson Prentice Hall. 906 p. ISBN 978-0-13-147139-9.

Hinz, T., Navarro-Guerrero, N., Magg, S. and Wermter, S., 2018. Speeding up the Hyperparameter Optimization of Deep Convolutional Neural Networks. International Journal of Computational Intelligence and Applications 17 (2): p. 1–15. Available from: https://doi.org/10.1142/S1469026818500086 [22.5.2019].

Hochreiter, S. and Schmidhuber, J., 1997. Long Short-Term Memory. Neural Computation 9 (8): p. 1735–1780.

Hänninen, M., Ylijoki, J., 2004. The Constitutive Equations of the APROS Six-equation Model. Espoo, VTT Technical Research Centre of Finland Ltd.

Van den Hof, P. M. J., 2012. System Identification - Data-Driven Modelling of Dynamic Systems, lecture notes. Available from: http://www.dcsc.tudelft.nl/~discsysid/ManuscrSysid_Feb2012.pdf [22.5.2019].

Ikonen E. and Najim K., 2002. Advanced process identification and control. New York, Marcel Dekker, Inc.: 310 p. ISBN 0-8247-0648-X.

Kingma, D. P. and Ba, J., 2014. Adam: A Method for Stochastic Optimization. ICLR 2015: 15 p. Available from: http://arxiv.org/abs/1412.6980 [22.5.2019].

Lappalainen, J., Blom, H. and Juslin, K., 2012. Dynamic Process Simulation as an Engineering Tool – A Case of Analysing a Coal Plant Evaporator. VGB Powertech 92 (1-2): p. 62–68. Available from: http://www.apros.fi/filebank/133-Dynamic_process_simulation_as_an_engineering_tool__A_case_of_analysing_a_coal_plant_evaporator.pdf [22.5.2019].

LeCun, Y., Bottou, L., Genevieve, B. O. and Muller, K-R., 1998. Efficient BackProp Neural Networks: Tricks of the Trade: p. 9–50. Available from: https://doi.org/10.1007/3-540-49430-8_2 [22.5.2019].

Leiviskä, K., 2013. Introduction to Experiment Design, lecture material. Oulu, University of Oulu.

Li, F., Zhang, J., Oko, E. and Wang, M., 2015. Modelling of a Post-Combustion CO_2 Capture Process Using Neural Networks. Fuel 151: p. 156–163. Available from: https://doi.org/10.1016/j.fuel.2015.02.038 [22.5.2019].

Liu, F., Quek, C. and Ng, G. S., 2005. Neural Network Model for Time Series Prediction by Reinforcement Learning. Proceedings of International Joint Conference on Neural Networks. Montreal, Que: p. 809–814.

Ljung, L. and Glad, T., 1994. Modeling of Dynamic Systems. New Jersey, Prentice Hall: 368 p. ISBN 978-013-59709-7-3.

Ljung, L., 1987. System identification: theory for the user. New Jersey, Prentice-Hall: 519 p. ISBN 0-13-881640-9.

May, R., Dandy, G. and Maier H., 2011. Review of Input Variable Selection Methods for Artificial Neural Networks. Artificial Neural Networks - Methodological Advances and Biomedical Applications, Prof. Suzuki, K. (Ed.). Intech. ISBN: 978-

953-307-243-2. Available from: http://www.intechopen.com/books/artificial-neural-networks-methodological-advances-and-biomedical-applications/review-of-input-variable-selection-methods-for-artificial-neural-networks [22.5.2019]

Nakisa, B., Rastgoo, M. N., Rakotonirainy, A., Maire, F. and Chandran, V., 2018. Long Short Term Memory Hyperparameter Optimization for a Neural Network Based Emotion Recognition Framework. IEEE Access 6: p. 49325–38. Available from: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8453798 [22.5.2019]

Ng A, 2019. Machine Learning – Advice for Applying Machine Learning, online course. Stanford University. Available from: https://www.coursera.org/learn/machine-learning [22.5.2019].

NIST/SEMATECH, 2012. E-Handbook of Statistical Methods. Available from: http://www.itl.nist.gov/div898/handbook/ [2.5.2019]

Orr, G. B. and Müller, K-R., 1998. Neural Networks Tricks of the Trade. Berlin Heidelberg, Springer-Verlag: 432 p. ISBN 978-3-540-49430-0.

Papacharalampous, G., Tyralis, H. and Koutsoyiannis, D., 2019. Comparison of Stochastic and Machine Learning Methods for Multi-Step Ahead Forecasting of Hydrological Processes. Stochastic Environmental Research and Risk Assessment 33 (2): p. 481–514. Available from: https://doi.org/10.1007/s00477-018-1638-6 [22.5.2019].

Ruder, S., 2016. An Overview of Gradient Descent Optimization Algorithms. Available from: http://arxiv.org/abs/1609.04747 [22.5.2019].

Shannon, C. E., 1949. Communication in the Presence of Noise. Proceedings of the IRE 37 (1): p. 10–21.

Srivastava, N. Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15: p. 1929–1958.

Stanislav, Z. 2003. Systems and Control. New York, Oxford University Press: 720 p.

ISBN 978-019-51501-1-7.

Stosch, M. V., Oliveira, R., Peres, J. and Azevedo, S. F., 2014. Hybrid Semi-Parametric Modeling in Process Systems Engineering: Past, Present and Future. Computers and Chemical Engineering 60: p. 86–101. Available from: http://dx.doi.org/10.1016/j.compchemeng.2013.08.008 [22.5.2019].

Sun, N-Z. and Sun, A., 2015. Model Calibration and Parameter Estimation: For Environmental and Water Resource Systems. New York, Springer-Verlag: 621 p. ISBN 978-1-4939-2322-9.

Sun, Y., Babovic, V. and Chan, E. S., 2010. Multi-Step-Ahead Model Error Prediction Using Time-Delay Neural Networks Combined with Chaos Theory. Journal of Hydrology 395 (1-2): p. 109-116. Available from: http://dx.doi.org/10.1016/j.jhydrol.2010.10.020 [22.5.2019].

Taieb, S. B., 2014. Machine Learning Strategies for Time Series Forecasting. Bryssel, Université libre de Bruxelles: 190 p. Available from: https://souhaib-bentaieb.com/pdf/2014_phd.pdf [22.5.2019].

Taieb, S. B. and Hyndman, R. J., 2012. Recursive and Direct Multi-Step Forecasting : The Best of Both Worlds. Monash University: 35 p. Available from: https://robjhyndman.com/papers/rectify.pdf [22.5.2019].

Taieb, S. B., Sorjamaa, A. and Bontempi, G., 2010. Multiple-Output Modeling for Multi-Step-Ahead Time Series Forecasting. Neurocomputing 73 (10–12): p. 1950–1957. Available from: http://dx.doi.org/10.1016/j.neucom.2009.11.030 [22.5.2019].

Tso, G. K. F. and Yau, K. K. W., 2007. Predicting electricity energy consumption: A comparison of regression analysis, decision tree and neural networks, Energy, 32 (9): p. 1761–1768. Available from: https://doi.org/10.1016/j.energy.2006.11.010 [22.5.2019].

Vapnik, V. N., 1999. An overview of statistical learning theory. IEEE Transactions on Neural Networks 10 (5): p. 988–999. Available from: https://ieeexplore.ieee.org/document/788640 [22.5.2019].

Venkatraman, A., 2017. Training Strategies for Time Series: Learning for Prediction, Filtering, and Reinforcement Learning. Pittsburgh, Carnegie Mellon University: 123 p. Available from: http://www.cs.cmu.edu/~arunvenk/papers/thesis.pdf [22.5.2019]

Wang, Y., Xie, Z., Hu, Q. and Xiong, S., 2018. Correlation Aware Multi-Step Ahead Wind Speed Forecasting with Heteroscedastic Multi-Kernel Learning. Energy Conversion and Management 163: p. 384–406. Available from: https://doi.org/10.1016/j.enconman.2018.02.034 [22.5.2019].

Wu, W. and Peng, M., 2017. A Data Mining Approach Combining K -Means Clustering With Bagging Neural Network for Short-Term Wind Power Forecasting. IEEE Internet of Things Journal 4 (4): p. 979–986: https://ieeexplore.ieee.org/document/7870674 [22.5.2019].

Wu, Y., Yuan, M., Dong, S., Lin, L. and Liu, Y., 2018. Remaining Useful Life Estimation of Engineered Systems Using Vanilla LSTM Neural Networks. Neurocomputing 275: p. 167–179. Available from: https://doi.org/10.1016/j.neucom.2017.05.063 [22.5.2019].

Xu, W., Peng, H., Zeng, X., Zhou, F., Tian, X. and Peng, X., 2019. A Hybrid Modelling Method for Time Series Forecasting Based on a Linear Regression Model and Deep Learning. Applied Intelligence. Available from: https://doi.org/10.1007/s10489-019-01426-3 [22.5.2019].

Yu, D. L., Gomm, J. B. and Williams, D., 2000. Neural Model Input Selection for a MIMO Chemical Process. Engineering Applications of Artificial Intelligence 13 (1): p. 15–23. Available from: https://doi.org/10.1016/S0952-1976(99)00046-9 [22.5.2019].

Yuan, X., Li, L. and Wang, Y., 2019. Nonlinear Dynamic Soft Sensor Modeling with Supervised Long Short-Term Memory Network. IEEE Transactions on Industrial Informatics. Available from: https://ieeexplore.ieee.org/document/8654687/ [22.5.2019].

APPENDICES

Appendix 1. Case study 2 – Hyperparameter optimisation runs of each model type and the best models by NRMSE on validation dataset in each.

Hyperparameter optimisation run	ARX-1	ARX-2	ARX-3	NARX-1	NARX-2	NARX-3	NARX-4	NARX-5	NARX-6	NARX-7	LSTM-1	LSTM-2	LSTM-3	LSTM-4	GRU-1	GRU-2	GRU-3	GRU-4
N models	962	2119	2119	2560	8127	5492	3147	3063	2213	5358	1461	508	527	2262	335	1531	267	556
N of input lags	1...5	1	1...5	1	1	1	1	1	1	1	1...6	1	1	1	1	1	1	1
N of output lags	1...5	1	1...5	1	1...2	1	1	1	1	1	1...6	1	1	1	1	1	1	1
N of layers	-	-	-	1...2	1...2	1...2	1	1	1	1	1...2	1	1	1	1	1	1	1
N of neurons (layer 1)	-	-	-	6...18	4...128	6...64	6...64	6...18	64...128	64...256	6...64	6...64	64...128	64...256	6...64	6...64	64...128	64...256
N of neurons (layer 2)	-	-	-	6...18	4...64	4...64	-	-	-	-	6...64	-	-	-	-	-	-	-
Learning rate	1e-4...5e-3	1e-4...5e-3	1e-4...5e-3	1e-4...1e-3	5e-5...4e-4	1e-4...2e-3	1e-4...2e-3	2e-4...1e-3	2e-4...1e-3	2e-4...2e-3	1e-4...2e-3	2e-4...1e-3	2e-4...1e-3	2e-4...2e-3	2e-4...1e-3	2e-4...1e-3	2e-4...1e-3	2e-4...2e-3
Learning rate decay	0	0	0	0	0...1e-6	0	0	0	0	0	0	0	0	0	0	0	0	0
Beta1	0,8...0,95	0,8...0,95	0,9	0,8...0,95	0,8...0,95	0,8...0,95	0,9	0,9	0,9	0,8...0,95	0,8...0,95	0,9	0,9	0,8...0,95	0,9	0,9	0,9	0,8...0,95
Beta2	0,8...0,999	0,8...0,999	0,999	0,9...0,99	0,9...0,99	0,9...0,99	0,999	0,999	0,999	0,8...0,999	0,8...0,999	0,999	0,999	0,8...0,999	0,999	0,999	0,999	0,8...0,999
N of epochs	10...200	10...150	10...500	50...150	3...400	10...500	10...300	10...100	100...250	50...500	10...500	10...400	200...500	50...500	10...300	10...1000	200...500	100...600
Batch size	8800	8800	8800	64...10000	10000	128...10000	8800	8800	8800	8800	128...8800	8800	8800	8800	8800	8800	8800	8800
N_dropout	-	-	-	0...3	1	0...3	0...3	0	1	0...1	0...3	0...3	1...3	0...1	0...3	0...3	1	0...1
Dropout rate	-	-	-	0,05...0,3	0,05...0,3	0,05...0,3	0,05...0,3	-	0,05...0,2	0,05...0,2	0,05...0,3	0,05...0,3	0,05...0,3	0,05...0,2	0,05...0,3	0,05...0,3	0,05...0,2	0,05...0,2
Maxnorm	-	-	-	0...5	0...1	0...5	0...5	0...5	0...5	0...5	0...5	0...5	0...5	0...5	0...5	0...5	0...5	0...5

Hyperparamter optimisation run	ARX-1	ARX-2	ARX-3	NARX-1	NARX-2	NARX-3	NARX-4	NARX-5	NARX-6	NARX-7	LSTM-1	LSTM-2	LSTM-3	LSTM-4	GRU-1	GRU-2	GRU-3	GRU-4
Number of models	962	2119	537	2560	8127	5492	3147	3063	2213	5358	1461	508	527	2262	335	1531	267	556
H2 feed in pressure	0,26	0,25	0,363	1,80	0,97	0,94	0,68	1,16	0,55	0,75	0,91	0,85	0,62	0,80	0,81	0,73	0,57	0,59
Product mass flow out	0,57	0,56	0,597	0,70	0,89	1,04	0,64	0,66	0,66	0,67	0,75	0,59	0,78	0,60	0,61	0,72	0,59	0,60
Product CH4-%	4,06	3,63	2,563	2,22	1,84	2,07	2,04	2,98	1,43	2,85	2,13	1,84	1,52	1,22	1,58	1,59	1,74	1,83
Product H2-%	1,95	2,03	1,942	1,79	1,22	1,29	1,15	2,47	1,18	1,24	1,23	1,18	1,13	1,34	1,36	1,30	1,11	1,21
Product CO2-%	2,75	2,50	1,838	1,97	1,59	1,81	1,75	1,89	1,73	2,86	2,14	2,11	1,65	1,63	1,61	1,77	1,83	2,02
Reactor average temp	2,64	2,91	2,955	3,41	2,31	2,45	2,91	2,64	3,09	2,57	2,34	1,72	1,26	1,37	2,18	1,18	1,33	1,45
Reactor max temp	0,53	0,54	0,54	1,57	0,84	1,83	2,04	1,06	1,23	1,47	1,89	1,47	0,95	0,99	1,73	1,18	1,22	1,38
Cooling mass flow out	3,10	3,08	1,92	1,53	2,19	0,74	0,49	1,86	0,61	0,45	1,42	0,98	0,61	0,97	0,96	0,79	0,30	0,48
Cooling steam enthalpy out	1,53	1,78	1,659	1,68	1,53	1,15	1,36	1,51	1,54	1,39	1,27	0,97	0,90	0,87	1,09	0,83	0,82	0,91
Cooling power	0,36	0,46	0,532	0,89	0,71	1,18	0,69	1,26	0,61	0,76	1,71	0,75	0,78	0,80	0,84	0,73	0,46	0,52
Recycling mass flow	0,29	0,27	0,542	1,62	0,98	0,82	0,75	0,97	0,60	0,59	0,72	0,56	0,61	0,88	0,62	0,61	0,49	0,67
Recycling compressor power	1,46	1,51	1,428	1,78	1,17	1,36	1,14	1,83	1,08	1,20	1,09	1,18	1,11	1,34	1,24	1,26	1,06	1,12
Average	1,62	1,63	1,406	1,75	1,35	1,39	1,30	1,69	1,19	1,40	1,47	1,18	0,99	1,07	1,22	1,06	0,96	1,06
Number of models in total		ARX	3618		NARX	29960		LSTM	4758		GRU	2689						