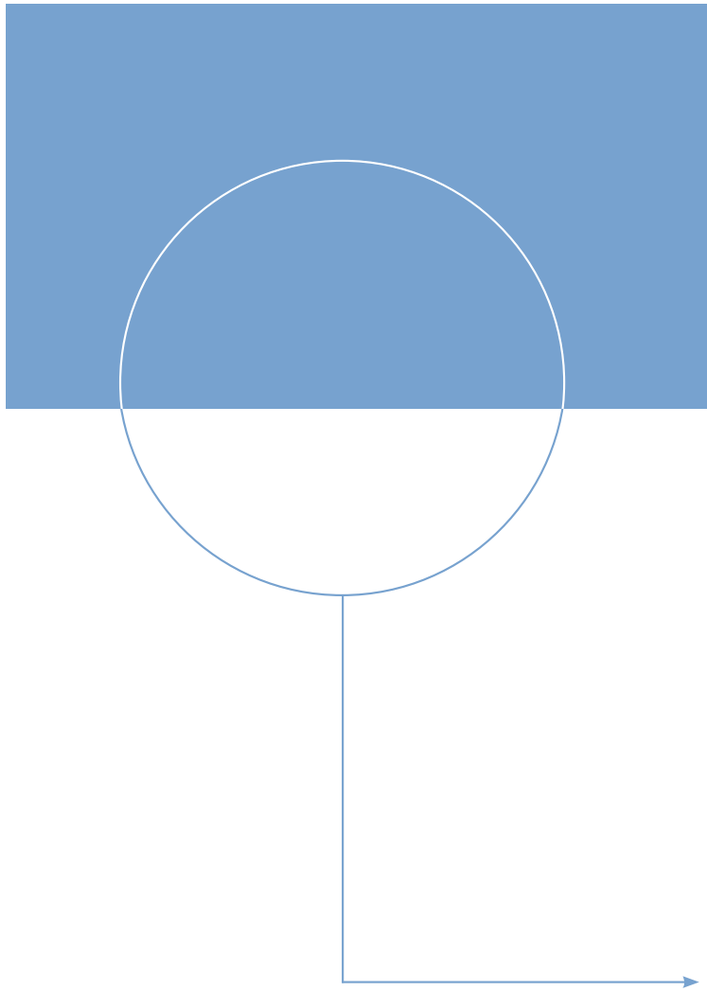


Master's thesis

2021

Ingeborg Kristine Eilertsen

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Engineering Cybernetics

Ingeborg Kristine Eilertsen

# Uncertainty estimation in autoregressive exogenous networks and nonlinear autoregressive exogenous neural networks

June 2021

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU

Norwegian University of
Science and Technology

# Uncertainty estimation in autoregressive exogenous networks and nonlinear autoregressive exogenous neural networks

**Ingeborg Kristine Eilertsen**

Submission date: June 2021
Supervisor: Prof. Jan Tommy Gravdahl
Co-supervisor: Dr. Esten Ingar Grøtli, Dr. Mark Haring, Dr. Signe Moe, Katrine Seel

Norwegian University of Science and Technology
Department of Engineering Cybernetics

# Abstract

Autoregressive exogenous (ARX) networks and nonlinear autoregressive (NARX) neural networks are reliant on a measure of uncertainty if they are going to be employed in safety critical tasks, where wrong decisions in the worst case scenario can lead to serious accidents. There are two main types of uncertainties which can be estimated in neural networks; epistemic uncertainty and aleatoric uncertainty. The former is caused by missing data-points in the training set the network is trained on. The latter, on the other hand, is caused by uncertainties in the network's inputs.

In this thesis, a method which propagates means and covariance through the network is proposed to estimate the aleatoric uncertainty. This method is tested on an ARX network in the combination with a Kalman filter which resets the estimated variance, where the system equations of the Kalman filter is given by the ARX network's model parameters. Further, is the aleatoric estimation method tested on a neural network and a NARX neural. Moreover, an epistemic uncertainty method which use dropout at test time to estimate uncertainty is applied to a NARX neural network.

The aleatoric uncertainty estimation method can be used to get good estimates of the uncertainty in ARX networks and neural networks. The method does also offer an estimate of the aleatoric uncertainty in a NARX neural network. However, this estimate is not a good estimate for the true aleatoric uncertainty in a NARX neural network. The epistemic uncertainty method estimates a higher uncertainty when predictions are made on data underrepresented in the training set. Furthermore, the uncertainty estimates scale with how represented the data on which the prediction is made is in the training set.

# Sammendrag

Autoregressive eksogene (ARX) nettverk og ikke-lineære autoregressive eksogene (NARX) nevrale nettverk er avhengige av et mål på usikkerhet hvis de skal brukes til sikkerhets kritiske oppgaver, der feil valg i det verste tilfellet kan føre til alvorlige ulykker. Det finnes to hovedtyper av usikkerheter man kan estimere i nevrale nettverk; modell-usikkerhet og data-usikkerhet. Den første kommer av manglende datapunkter i treningssettet som nettverket trenes på, og den andre er forårsaket av usikkerhet knyttet til nettverkets innganger.

I denne masteroppgaven er foreslås det en data-usikkerhetsestimeringsmetode som propagerer forventningsverdier og kovarianser gjennom nettverket. Denne metoden er testet på et ARX nettverk i kombinasjon med et Kalman filter, som resetter den estimerte variansen. Kalman filteres systemligninger er bestemt av ARX nettverkets modellparameter. Videre vil denne data-usikkerhetsestimerings metoden bli teste på et nevralt nettverk og et NARX nevralt nettverk. En modell-usikkerhetsestimeringsmetode vil også bli teste på et NARX nevralt nettverk. Denne bruker dropout til å estimer modell-usikkerheten.

Data-usikkerhetsestimeringsmetoden kan brukes til å oppnå gode estimater for usikkerheten i ARX nettverk og nevrale nettverk. Metoden gir også et estimat for data-usikkerheten i et NARX nevralt nettverk, men dette estimatet er ikke et godt estimat på den reelle data-usikkerheten i nettverket. Model-usikkerhetsestimeringsmetoden estimerer høyere usikkerhet for prediksjoner som er underrepresenterte i treningssettet. Videre, skalerer den estimerte usikkerhet med hvor representert dataen som prediksjonen er utført på er i treningssettet.

# Acknowledgments

I want to thank my supervisors Professor Jan Tommy Gravdahl, Dr. Esten Ingar Grøtli, Dr. Mark Haring, Dr. Signe Moe, and Katrine Seel for valuable guidance throughout this master thesis.

I also want to thank my family for their support and time spent proofreading.

# Contents

# 1 Introduction

## 1.1 Background

This section is a rewritten version of section *1.1 Background* in Eilertsen 2020.

Machine learning is a field of research which has taken enormous strides in the last decades. The majority of this progress is due to increased availability of data and improvements in computing technology. One area of machine learning is regression which entails approximating relations between input-output pairs. Regression can be split into two parts, linear regression and nonlinear regression. Deep neural networks are used to solve nonlinear regression problems. These networks are composed of one or several layers which each contain linear transformations and nonlinear functions. Linear regression models are used to solve linear regression problems. These models contains one layer which is linear.

A nonlinear autoregressive exogenous (NARX) neural network is a special type of deep neural network which approximates nonlinear dynamical systems
(Narendra and Parthasarathy 1990). A NARX neural network performs one-step ahead predictions of the dynamical system based on the system's current and previous inputs, and the network's previous predictions. Thus, a NARX neural network is recurrent as the network's previous predictions are used to obtain the next prediction. Figure

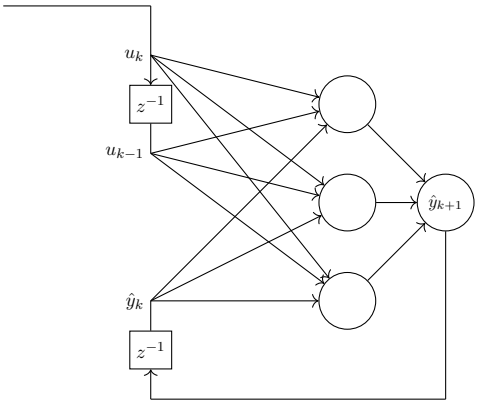



Figure 1.1: Graphical illustration of a NARX neural network. Where one previous input together with the current input, and the previous output is used to make the next prediction.

1.1 shows a graphical illustration of a NARX neural network. NARX neural networks have many applications, some examples are compensations of nonlinearities in dynamical systems, economic forecasting, solar forecasting and weather forecasting (Tavares, Abreu, and Aguirre 2020) (Tang 2020) (Alanazi, Mahoor, and Khodaei 2017) (Rahimi, Mohd Shafri, and Norman 2018). An autoregressive exogenous (ARX) network is a NARX neural network which consist of only one linear layer. These networks are used to approximate linear dynamical systems.

One significant weakness of traditional machine learning models which solves regression problems, are that they do not offer a measure of uncertainty associated with their predictions. When used in safety critical tasks, where wrong decisions can be detrimental and might lead to fatalities, injuries, material damage, or stop of production, a measure of uncertainty can be used to determine if the models prediction should be used or if another mechanism should take over.

In machine learning there are two leading types of uncertainties which can be estimated, aleatoric and epistemic uncertainty (Gal 2016). The former is the uncertainty caused by uncertainty in the network's inputs, uncertainty in yields uncertainty out. Epistemic uncertainty, on the other hand, is the uncertainty stemming from missing input-output pairs in the data set the model is trained on. There is a higher level of uncertainty associated with predictions made on data which is seen less during training.

In recent years several different methods for estimating uncertainty in deep neural networks have been proposed. Lakshminarayanan, Pritzel, and Blundell 2017 estimates epistemic uncertainty by performing Monte Carlo sampling on an ensemble of different networks. Monte Carlo sampling consists of collecting independent samples from a distribution, and approximating the mean and variance of the distribution with these samples. This method's downside is that several networks have to be trained, which might be unsuitable and impractical for large networks. The advantages of this method are that traditional deep learning techniques can be used. On the other hand, Gal and Ghahramani 2015 estimate epistemic uncertainty by applying dropout (Srivastava et al. 2014) at test time. Dropout consists of removing some of the network's units at random for each forward pass, and in the next forward pass the units are included in the network and some other units are removed. The uncertainty is found by Monte Carlo sampling; thus, this technique's drawback is the time it takes to obtain the samples. The method's strengths are that it is easy to implement and there is no need to change network architecture. The method can also be applied on already trained networks regardless of being trained with dropout. Diversely Postels et al. 2019 proposes a method for estimating epistemic uncertainty which does not require sampling. The method is based on error propagation where errors are equivalent to variances (Taylor 1996). Noise is injected in the network with a noise layer, which can either be a dropout layer or a batch normalization layer (Ioffe and Szegedy 2015). The injected noise is regraded as errors on the output of the noise layers. Then these errors are propagated to the output of the network. When training a neural network with noise, the training loss will be greater when the errors on the outputs are greater. Therefore, the network will indirectly minimize

errors since the aim of training is to minimize the training loss. The main advantage of this method is that it does not involve sampling.

Gast and Roth 2018 propose a method for estimating aleatoric uncertainty by utilizing assumed density filtering (ADF) to propagate the input noise through the network (Boyen and Koller 1998),(Maybeck 1979), (Lauritzen 1992),(Opper and Winther 1999). The network propagates intermediate estimates and variances. It is assumed that the output of each network layer is independent and that the network's inputs are independent. The main advantage of this method is that it only requires a small change in network architecture. The method's weakness is that it assumes independence, which is generally not the case.

Another paradigm for uncertainty estimation is Bayesian neural networks(MacKay 1992), (Geoffrey E. Hinton and Neal 1995), where the network's model parameters are distributions, in contrast to traditional neural networks where the model parameters are fixed. Bayesian neural networks typically estimate either aleatoric uncertainty or epistemic uncertainty. Kendall and Gal 2017 propose a Bayesian neural network which estimates both aleatoric and epistemic uncertainty. The main disadvantage of Bayesian neural networks is that traditional deep learning techniques can not be used.

## 1.2 Problem Formulation

This thesis is concerned with estimating uncertainty in regression tasks in machine learning, with a focus on uncertainty estimation in NARX neural networks and aleatoric uncertainty estimation in ARX networks.

This thesis aims to improve the existing aleatoric uncertainty estimation method ADF proposed by Gast and Roth 2018 to obtain an estimate of aleatoric uncertainty which is closer to the true uncertainty. By assuming that the network's inputs are dependent and that the output from each network layer is dependent, as opposed to the original method where this is assumed independent. The altered method will be tested on the following systems:

1. ARX network in combination with a Kalman filter to reset the uncertainty estimate
2. Feedforward neural network
3. NARX neural network

Furthermore, will the thesis try to answer the following question: Is the aleatoric uncertainty in a neural network the same as the aleatoric uncertainty in the system it approximates?

Moreover, estimating epistemic uncertainty with dropout and Monte Carlo sampling (Gal and Ghahramani 2015) will be implemented and evaluated with regards to uncertainty estimation capabilities in a NARX neural network. This method will be denoted

by MC-dropout.

These methods have been chosen because they only require minimal network architecture changes, and existing deep learning techniques can be used; most importantly, the networks training process is unchanged. Moreover, these methods can be employed on already trained networks with minimal modifications. To the author's best knowledge, these two methods have never previously been applied to NARX neural networks.

## 1.3 Outline Of This Thesis

This thesis is organized as following:

Chapter 2 gives an introduction to machine learning, where supervised learning with a focus on regression, linear regression and deep feedforward networks are presented. Moreover, ARX and NARX models are presented together with ARX networks and NARX neural networks. Furthermore, uncertainty in neural networks is presented. Followed by a description of Monte Carlo sampling. Then the uncertainty estimation methods, ADF to estimate aleatoric uncertainty and MC-dropout to estimate epistemic uncertainty are presented. Finally, the Kalman filter is described.

Next in Chapter 3 the extended version of ADF is presented. Then, the output mean of a rectified linear unit activation function is given when the input is multivariate Gaussian distributed. Next, the output covariance of the same function is derived when the input is multivariate Gaussian distributed. Further, the uncertainty estimations methods implementation details are described, together with their training details and the systems they are tested on.

In Chapter 4 experiments and their results will be presented.

Chapter 5 the results from the experiments will be discussed, and further work will be presented.

In Chapter 6 a conclusion is given.

# 2 Theory

This chapter gives an overview of the theory needed in this thesis. First machine learning will be presented in section 2.1 with a focus on regression tasks, specifically linear regression which is presented in section 2.1.2 and deep feedforward networks presented in section 2.1.3. Further, autoregressive exogenous models and nonlinear autoregressive exogenous models are presented together with autoregressive exogenous networks and nonlinear autoregressive exogenous neural networks in section 2.2. Then, in section 2.3 uncertainty in neural networks is described. Further, assumed density filtering for estimating aleatoric uncertainty is presented, in section 2.4. Next, in section 2.5 a description of Monte Carlo Sampling is given. Then, MC-dropout to estimate epistemic uncertainty is presented in section 2.6. Finally, in section 2.7 the Kalman filter will be described.

## 2.1 Machine Learning

Machine learning is a subset of artificial intelligence, consisting of a collection of algorithms that learn from experience. Learning from experience means that the algorithm improves its performance measure in accomplishing a task when it has more experience (Mitchell 1997). The performance measure is different for different tasks, depending on what the algorithm is supposed to learn.

There are three main classes of machine learning: unsupervised learning, reinforcement learning, and supervised learning. In unsupervised learning, the experience is a dataset composed of features. An unsupervised learning algorithm wants to find the patterns in the dataset. Reinforcement learning algorithms interact with an environment. The algorithm's experience is the feedback it gets from the environment based on decisions the algorithm makes. In supervised learning, the experience is a dataset consisting of inputs and outputs of a system. The learning algorithm wants to learn the relation between the inputs and the outputs. Tasks performed by a supervised learning algorithm can be separated into two main categories, classification, and regression. Classification consists of mapping the input into distinct categories. For example, if the input is an image containing either a cat or a dog, the classification learning algorithm should determine if the image depicts a cat or a dog. Regression, on the other hand, consists of mapping the input to a numerical value.

This section will give an overview of supervised learning in the context of regression tasks. We are starting with an outline of supervised learning in section 2.1.1. Further

two regression algorithms will be presented, beginning with linear regression in section 2.1.2 and continuing with deep feedforward networks in section 2.1.3.

### 2.1.1 Supervised learning - Regression

This section is based on Goodfellow, Bengio, and Courville 2016. A supervised learning regression algorithm wants to find the relation between the inputs and the outputs of an unknown system. Given a dataset with $N$ input-output pairs from a system, $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$, where $\mathbf{x}_n$ and $\mathbf{y}_n$ are vectors or scalars, the algorithm wants to best model the relation between the inputs and outputs.

The data in the dataset $\mathcal{D}$ comes from a data-generating process. There exists a hypothetical probability distribution over the data-generating process that describes how the data in the data-generating process is distributed. The dataset consists of independently drawn samples from this distribution. So the dataset is said to be empirically distributed. Thus, the distribution over the dataset is the observed distribution over the data-generating process. The overarching of the machine learning algorithm is to learn the data-generating distribution based on the empirically distribution. The distribution that the algorithm learns is called the model distribution.

### 2.1.2 Linear Regression

One supervised machine learning algorithm is linear regression. This section is based on Goodfellow, Bengio, and Courville 2016. Given a dataset with $N$ input-output pairs, $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$, where $\mathbf{x}_n$ and $\mathbf{y}_n$ for $n \in \{1, .., N\}$ where $N$ is positive, are vectors or scalars, the algorithm wants to best approximate the relation between the inputs and outputs. It is assumed that this relationship is linear and can be described by $\mathbf{y}_n = \mathbf{f}^*(\mathbf{x}_n) + \varepsilon_n \; n \in \{1, .., N\}$, where $\mathbf{f}^*$ is an unknown linear function and $\varepsilon_n$ is noise. The algorithm wants to find a linear function on the from $\hat{\mathbf{y}_n} = \mathbf{f}(\mathbf{x}_n) = \mathbf{W}\mathbf{x}_n + \mathbf{b}$ that best models $\mathbf{f}^*$. The matrix $\mathbf{W}$ and the vector $\mathbf{b}$ are called the weight matrix and the bias vector respectively, and are the model parameters. The model parameters are often gathered in a model parameter vector, $\boldsymbol{\theta}$.

The dataset $\mathcal{D}$ can be written on matrix form as $\mathbf{X}$ and $\mathbf{Y}$, where $\mathbf{x}_n$, with $n \in \{1, .., N\}$ are the rows of $\mathbf{X}$, and similarly $\mathbf{y}_n$, with $n \in \{1, .., N\}$ are the rows of $\mathbf{Y}$. The dataset $\mathcal{D}$ can be split up into two datasets. Where the first dataset is used for training called the training set, and the second dataset is called the test set and is used for testing. These two data sets can be denoted by $\mathcal{D}^{(train)} = \{\mathbf{X}^{(train)}, \mathbf{Y}^{(train)}\}$ and $\mathcal{D}^{(test)} = \{\mathbf{X}^{(test)}, \mathbf{Y}^{(test)}\}$. The training set is used to learn the model parameters, and the test set is used to evaluate the model's performance.

To find the model distribution that most closely resembles the data-generating distribution one can maximize the likelihood of the conditional probability

$$P(\mathbf{Y}^{(train)} \mid \mathbf{X}^{(train)}; \boldsymbol{\theta}) \tag{2.1}$$

with respect to $\boldsymbol{\theta}$. This can be expressed as

$$\boldsymbol{\theta}_{ML} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, P(\mathbf{Y}^{(train)} \mid \mathbf{X}^{(train)}; \boldsymbol{\theta}) \tag{2.2}$$

where $\boldsymbol{\theta}_{ML}$ is $\boldsymbol{\theta}$ which maximizes the likelihood of (2.1). It can be shown that maximizing this likelihood is equivalent to minimizing the mean square error between the outputs in the training set and the linear regressions output, which is given by $\mathbf{X}^{(train)}\boldsymbol{\theta}$, which can be expressed as the following

$$MSE = \frac{1}{m}\left\|\mathbf{X}^{(train)}\boldsymbol{\theta} - \mathbf{Y}^{(train)}\right\|_2^2 \tag{2.3}$$

assuming that there are $m$ examples in the training dataset. It can be shown that minimizing the mean squared error with respect to $\boldsymbol{\theta}$ yields

$$\boldsymbol{\theta} = ((\mathbf{X}^{(train)})^T\mathbf{X}^{(train)})^{\dagger}(\mathbf{X}^{(train)})^T\mathbf{Y}^{(train)} \tag{2.4}$$

where † denotes the Moore–Penrose inverse (Harville 1997).

### 2.1.3 Deep forward networks

One type of machine learning model is neural networks, which are most commonly used for supervised learning. This section is based on section 2.1.1 *Deep forward networks* in Eilertsen 2020. The majority of this section is based on Goodfellow, Bengio, and Courville 2016. A deep feedforward network's objective is to approximate an unknown function $\mathbf{f}^*$ given a dataset containing input-output pairs of the unknown function. The inputs are called examples, and the outputs are called labels. They are denoted by $\mathbf{x}$ and $\mathbf{y}$ respectively, $\mathbf{y} = \mathbf{f}^*(\mathbf{x})$.

A deep feedforward network defines a mapping $\mathbf{y} = \mathbf{f}(\mathbf{x}; \boldsymbol{\theta})$. $\mathbf{x}$ is the network's input, $\mathbf{y}$ is its output, and $\boldsymbol{\theta}$ is a parameterization of the network. This map can be decomposed into several maps

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}^{(L)}(\mathbf{f}^{(L-1)}(...(\mathbf{f}^{(1)}(\mathbf{x}; \theta)))) \tag{2.5}$$

where each $\mathbf{f}^{(l)}$, $l \in \{1, .., L\}$, is a layer of the network. The number of layers is the depth of the network. The L-th layer is called the output layer, and the first layer is called the input layer. The rest of the layers are the hidden layers, and they can be expressed as

$$\mathbf{z}^{(l)} = \mathbf{f}^{(l)} = \phi(\mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}) \tag{2.6}$$

where $\phi$ is an activation function which usually is non-linear. Examples of commonly used activation functions are the rectified linear unit activation (ReLU) function which is defined as

$$ReLU(x) = max\{0, x\} \tag{2.7}$$

the hyperbolic tangent function and the sigmoid function. If the functions input is a vector the functions are applied element-wise. $\mathbf{z}^{(l-1)}$ is a vector of outputs from the

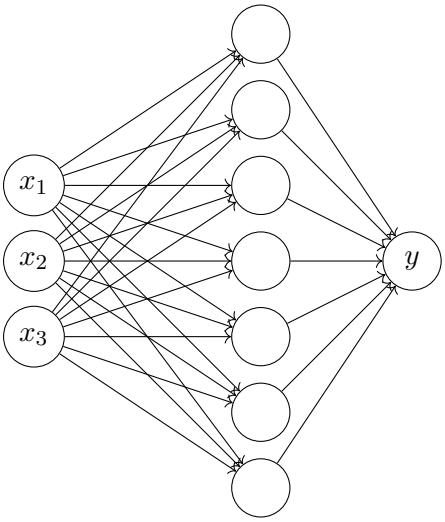



Figure 2.1: Graphical illustration of a deep feedforward network with three inputs, one output and one hidden layer with seven units. Each node represents a unit.

previous layer. $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are trainable parameters, where $\mathbf{W}^{(l)}$ is the weight matrix which connects the outputs from layer $l-1$ with the $l$-th layer, and $\mathbf{b}^{(l)}$ is the layer's bias vector. $\mathbf{z}^{(l)}$ is also referred to as an activation.

Each hidden layer again consists of units. The number of units in a layer is called the width of the layer. Each unit $i$ in a given layer $l$ can be expressed as

$$z_i^{(l)} = \phi(w_i^{(l)}\mathbf{z}^{(l-1)} + b_i^{(l)}) \tag{2.8}$$

where $b_i$ is the unit's bias, and $w_i^{(l)}$ are the weights connecting the outputs from the previous layer with the unit.

The input and output layers are different from the hidden layers because they do not normally contain activation functions. A deep feedforward network can be visualized as a graph depicted in figure 2.1, where each unit is represented by a node.

Neural networks extends linear regression which is described in section 2.1.2. Linear regression models are neural network which have one linear layer connecting the input with the output.

Now that the basics of deep feedforward networks have been presented, the training process can be explored. As previously mentioned, the trainable parameters are the weight matrices connecting each layer and each layer's bias vector. Let $\boldsymbol{\theta}$ contain these parameters. The training aims to find the $\boldsymbol{\theta}$ which makes the network $\mathbf{f}$, best approximate the unknown function $\mathbf{f}^*$. A measure for how good this approximation is, is the mean

square error (MSE) given by

$$J = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.9}$$

where $\mathbf{y}_i$ is the label and $\hat{\mathbf{y}}_i$ is the network's corresponding prediction. The process of obtaining the estimate $\hat{\mathbf{y}}$ is known as forward propagation or a forward pass. An input $\mathbf{x}$ is given to the network, and is then propagated through the layers of the network. $J$ is referred to as a cost function. The model parameters can be updated by gradient descent

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J \tag{2.10}$$

where $\alpha$ is the learning rate.

There are three main methods for finding the gradients, namely batch gradient descent, stochastic gradient descent and mini-batch gradient descent. The first method consists of calculating the cost function on the whole dataset. In contrast, the second method calculates the cost function on one sample of the dataset at a time. The third method is a combination of the two others, where the cost function is calculated on a subset of the dataset. Mini-batch gradient descent usually converges faster than the two other methods (Ruder 2017).

One way of obtaining the gradient of $J$ with respect to $\boldsymbol{\theta}$ is known as back-propagation (Rumelhart, G. E. Hinton, and Williams 1986). Back-propagation utilizes the chain rule of calculus and computational graphs to calculate the gradient. The method exploits the fact that a lot of the derivatives share terms.

When the gradients have been found, $\boldsymbol{\theta}$ can be updated. This is done with an optimizer. An example of an optimizer is equation (2.10). Other optimizeres may extend the second term of equation (2.10) to include more complex elements such as momentum and adaptive learning rates to achieve faster convergence.

When training a network, some data is removed from the dataset and kept aside for testing. The performance, for example measured with the MSE described by Equation (2.9), on the data kept aside is called the generalization error, which is a measure of how well the network performs on unseen data. The network's goal is to minimize the generalization error, and by doing that, approximating the function $\mathbf{f}^*$. It is called overfitting when the network performs well in training but fails to generalize. Overfitting occurs because the network is taking into account noise in the training data and is modeling noise instead of the underlying process that has generated the data. The network is modeling a higher-order model than the underlying data. Regularization is a collection of techniques that aim to prevent overfitting. Two such techniques are L2-regularization and dropout.

### L2 regularization

L2-regularization is also known as ridge regression. It forces the weights to be small by adding the following term to the cost function described by Equation (2.9)

$$\lambda \sum_{w_i \in \theta} w_i^2 \tag{2.11}$$

where $w_i$ are the network's model parameters and $\lambda$ is a positive tuneable regularization parameter.

### Dropout

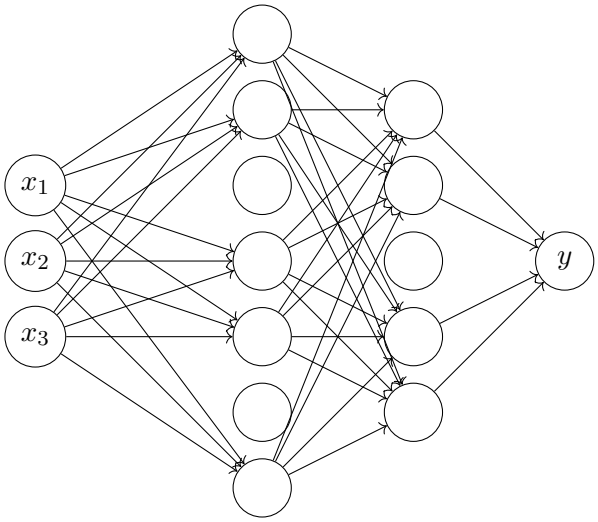



Figure 2.2: Graphical illustration of dropout in a neural network with three input values, two hidden layers, and one output. Dropout is applied at every network layer. Here three units are not connected to the units in the previous and next layer.

Dropout is another regularization method (Srivastava et al. 2014). It consists of ignoring some of the network's units during training. For each forward pass and corresponding backward pass, a percentage of the units are ignored, essentially removing some of the network's units. Which units that are removed is random. In the next forward and backward pass other units are ignored. During validation and testing, all units are active. The percent of which are ignored is called the dropout rate, denoted by $p$. Usually, it is set to between 20% - 50% for each layer. Dropout is graphically illustrated in figure 2.2. Here three units are not connected to the units in the previous and next layer.

When dropout is active, the output of each layer is scaled by $\frac{1}{(1-p)}$ where $p$ is the dropout rate for the layer. This is done to mitigate the loss of magnitude when units are ignored.

Mathematically dropout for a layer can be expressed as

$$\mathbf{z}^{(l)} = \phi(\mathbf{W}^{(l)}\mathbf{r}^{l} \circ \mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}) \tag{2.12}$$

where $\mathbf{r}^{l}$ is a vector consisting of independent Bernoulli random variables, each of which has probability $p$ of being zero, $r_j^{(l)} \sim \text{Bern}(1-q)$, where $p = 1 - q$ and $\circ$ denotes the element-wise product, which takes two matrices with the same dimensions and performs element-wise multiplication, to produce a new matrix with the same dimensions.

## 2.2 Autoregressive Exogenous Model (ARX)

This section is based on section 2.2 *Nonlinear autoregressive exogenous model (NARX)* in Eilertsen 2020. An autoregressive exogenous model (ARX) is a model that relates the next value of a time series to the past and current value of the time series and the past and current values of a driving input of system. The system can either be linear or nonlinear. When the system is nonlinear it is called a nonlinear autoregressive exogenous model (NARX).

When the system is linear the discrete ARX model with a scalar output can be expressed as

$$y_{k+1} = \sum_{i=0}^{m_y} a_i y_{k-i} + \sum_{i=0}^{m_u} b_i u_{k-i} + \varepsilon_{k+1} \tag{2.13}$$

where $a_i$ and $b_i$ are constants, and $y_{k+1}$ is the next value of the time series, and $y_k, ..., y_{k-m_y}$ are the current and previous values of the time series, $m_y$ describes how many previous steps of the time series which are used to find the current value.
$u_k, .., u_{k-m_u}$ are the current and previous driving input, $m_u$ indicates how many previous inputs are used. And $\varepsilon_{k+1}$ is noise.

When the system is nonlinear the NARX model can be expressed as the following

$$y_{k+1} = f(u_k, ..., u_{k-m_u}, y_k, ..., y_{k-m_y}) + \varepsilon_{k+1}$$

where $f$ is an unknown nonlinear function, and the rest of the variables are the same as the ARX model.

### 2.2.1 ARX network

An ARX model can be approximated with linear regression, described in section 2.1.2, where one or several previous predictions are used as inputs in the next prediction together with one or several of the inputs of the system that is approximated. An ARX model approximated by linear regression will be referred to as an ARX network.

Figure 2.3 shows a graphical representation of an ARX network. The network visualized has the system's current input, $u_k$, the previous system input, $u_{k-1}$, and two previous

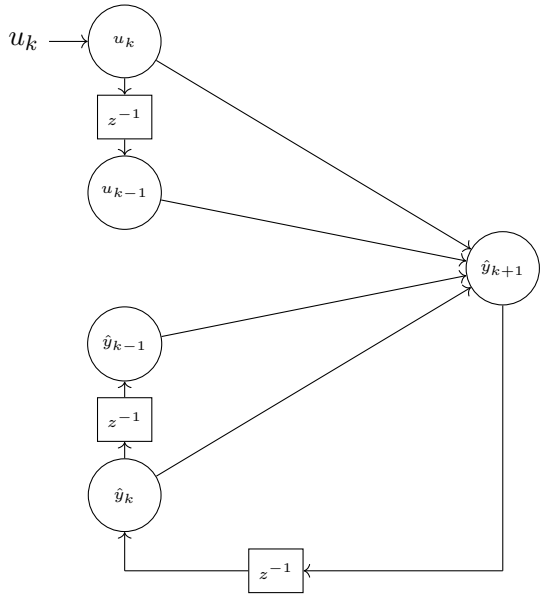



Figure 2.3: Graphical illustration of an ARX network with one output. And the current and a previous system input and two previous outputs as network inputs.

model predictions, $\hat{y}_k$ and $\hat{y}_{k-1}$ as model inputs. These inputs are used to predict the systems next output, $\hat{y}_{k+1}$. The edges from the inputs to the output represents the multiplication of the inputs with the model parameters, $\boldsymbol{\theta}$.

### 2.2.2 NARX neural network

A NARX model can be approximated by a NARX neural network (Narendra and Parthasarathy 1990). A NARX neural network is a recurrent network that performs one-step ahead predictions of a discrete non-linear system based on the system's current and previous inputs and previous outputs. The network is recurrent because the network's current output is fed back as a network input in the next time step, so that the network can perform consecutive one-step ahead predictions.

Figure 2.4 is a graphical illustration of a NARX neural network where one previous input, the current input and two previous outputs are used to estimate the next output. The $u_k, .., u_{k-m_u}$ denotes the systems and networks inputs where $m_u$ are the number of previous inputs used to obtain the prediction. In the figure $m_u$ is one. $u_k$ is the system's current input. Similarly, the $\hat{y}_k, ..., \hat{y}_{k-m_y}$ denotes the previous outputs from the network, where $m_y + 1$ is the number of previous outputs which are used as inputs to the network to obtain the prediction. In the figure, $m_y$ is equal to one. $\hat{y}_k$ is the output from the previous time step. $\hat{y}_{k+1}$ denotes the network's prediction.

The training of a NARX neural network is usually executed on a feedforward network. This is feasible when time-series data of the systems inputs and outputs is available.

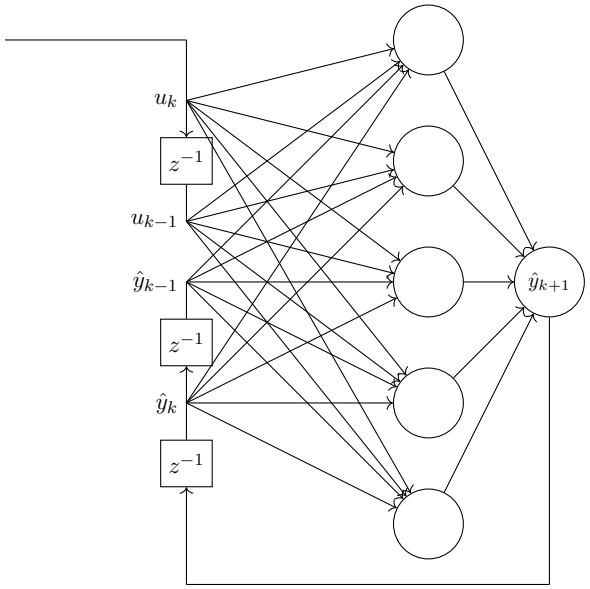



Figure 2.4: Graphical illustration of a NARX neural network with one hidden layer with 5 units and one output. The network uses one previous input and the current input and two outputs as inputs.

Training it in this manner is favoured because then mini-batch gradient decent can be used. If it where to be trained in its recurrent form, stochastic gradient descent has to be used to update the model parameters. The network is converted to a recurrent network after training.

## 2.3 Uncertainty In Neural Networks

The uncertainty associated with a deep neural network's predictions is affected by two main types of uncertainties; epistemic and aleatoric uncertainty (Gal 2016). The former is caused by missing data in the data the network is trained on, and the latter is induced by uncertainties affecting the network's input.

### 2.3.1 Epistemic Uncertainty

Epistemic uncertainty is also known as model uncertainty. It is the uncertainty due to missing data points in the training set; specific data from the input and output space are underrepresented or are not included in the training set at all. It is not always possible to represent all potential edge cases in the training data because those situations occur so infrequently that it is impossible to record them or it is infeasible to gather more observations due to cost.

A neural network can only model the data it is trained on. Thus, situations that do not

frequently appear in training data will have a higher degree of epistemic uncertainty than data that often occur. The epistemic uncertainty usually decreases when the amount of training data is increased. However, just adding more data to the training set does not always reduce the model uncertainty. If the training data added is similar to the data that is already in the training set, the epistemic uncertainty will not decline. If data that is dissimilar is added, the epistemic uncertainty will decrease.

### 2.3.2 Aleatoric Uncertainty

Aleatoric uncertainty, also called data uncertainty, is the uncertainty caused by uncertainty in the inputs. All observations are corrupted by noise because all sensor measurements are affected by noise. Hence, it is impossible to obtain an exact measurement. Some characteristics of this noise is often known and provided by the sensor manufacturer. This noise affects the uncertainty associated with the network prediction.

There are two types of aleatoric uncertainty, homoscedastic and heteroscedastic aleatoric uncertainty (Kendall and Gal 2017). Homoscedastic aleatoric uncertainty, the uncertainties associated with the inputs, are constant for all predictions, but they might be different for the different inputs used for each prediction. Different types of sensors have different levels of noise associated with them. Heteroscedastic aleatoric uncertainty, on the other hand, is when the uncertainties associated with the inputs are not constant for all predictions.

### 2.3.3 Uncertainty In NARX Neural Networks

A NARX neural network or any neural network where the network's output is used as network inputs in subsequent predictions has heteroscedastic aleatoric uncertainty. The data uncertainty affecting the current input depends on the uncertainty associated with a number of the network's previous predictions.

## 2.4 Assumed Density Filtering (ADF)

This section is a rewritten version of section *ADF - propagation* in Eilertsen 2020. Assumed density filtering is a Bayesian inference method, where an exact update step is performed, and then the posterior is approximated by a tractable distribution, $q(x)$ (Boyen and Koller 1998),(Maybeck 1979),(Lauritzen 1992),(Opper and Winther 1999).

Gast and Roth 2018 proposes a method for estimating data uncertainty by applying ADF to a deep neural network. Traditional units are replaced by probabilistic units, which instead of propagating intermediate values, propagates probability distributions, or more specifically; means and variances. Since this method uses ADF to propagate the input noise through the network it will be denoted by ADF.

The joint density over all activations in a deterministic deep neural network is given by

$$p(\mathbf{z}^{(0:l)}) = p(\mathbf{z}^{(0)}) \prod_{i=1}^{l} p(\mathbf{z}^{(i)} \mid \mathbf{z}^{(i-1)}) \tag{2.14}$$

$$p(\mathbf{z}^{(i)} \mid \mathbf{z}^{(i-1)}) = \delta[\mathbf{z}^{(i)} - \mathbf{f}^{(i)}(\mathbf{z}^{(i-1)})] \tag{2.15}$$

where $\delta[\cdot]$ is the Dirac delta and $\mathbf{f}^{(i)}$ is network layer $i$. The input in deterministic networks, $p(\mathbf{z}^{(0)})$ is Dirac delta distributed, thus they are assumed to be noise free. In general inputs are not noise free, especially if they are measurements of a physical process or asset. Therefore it is assumed that the probabilistic network's inputs are corrupted by Gaussian white noise. This can be expressed as

$$p(\mathbf{z}^{(0)}) = \prod_{j} \mathcal{N}(z_j^{(0)} \mid x_j, \sigma_j^2) \tag{2.16}$$

where $x_j$ and $\sigma_j^2$ is the mean and the variance of input $j$. The aim of the probabilistic network is to propagate this uncertainty to the output, and find $p(\mathbf{z}^{(0:l)})$. This distribution is intractable, hence it is approximated by ADF. Which can be described by

$$p(\mathbf{z}^{(0:l)}) \approx q(\mathbf{z}^{(0:l)}) = q(\mathbf{z}^{(0)}) \prod_{i=1}^{l} q(\mathbf{z}^{(i)}) \tag{2.17}$$

where $q(\mathbf{z}^{(0)}) = p(\mathbf{z}^{(0)})$ is the network's input. It is assumed that every $q(\mathbf{z}^{(i)})$ is independently Gaussian distributed. This can be expressed as

$$q(\mathbf{z}^{(i)}) = \prod_{j} \mathcal{N}(z_j^{(i)} \mid \mu_j^{(i)}, v_j^{(i)}) \tag{2.18}$$

where $\mu_j$ is the activation value for the j-th unit in the layer, and $v_j$ is the variance associated with that activation value.

The noise which is affecting the inputs is propagated through the network. After each layer $q(\mathbf{z}^{(i)})$ is calculated and passed to the next layer, a layer, $\mathbf{f}^{(i)}$ takes a distribution $q(\mathbf{z}^{(i-1)})$ as an input and transforms it into a joint probability density distribution, which is expressed as

$$p(\mathbf{z}^{(i)} \mid \mathbf{z}^{(i-1)}) q(\mathbf{z}^{(i-1)}) \tag{2.19}$$

This distribution is not necessarily independently Gaussian distributed and can be a complex form. ADF approximates this distribution to be

$$\tilde{p}(\mathbf{z}^{(0:i)}) = p(\mathbf{z}^{(i)} \mid \mathbf{z}^{(i-1)}) \prod_{j=0}^{i-1} q(\mathbf{z}^{(j)}) \tag{2.20}$$

ADF then finds $q(\mathbf{z}^{(i)})$ by minimizing the Kullback–Leibler divergence (Kullback and Leibler 1951)

$$q(\mathbf{z}^{(i)}) = \underset{\tilde{q}(\mathbf{z}^{(0:i)})}{\operatorname{argmin}} D_{\mathrm{KL}} \left( \tilde{p}(\mathbf{z}^{(0:i)} \mid\mid \tilde{q}(\mathbf{z}^{(0:i)}) \right) \tag{2.21}$$

Minimizing the Kullback-Leibler divergence is the same as matching the expectation and the variance of the two distributions (Minka and Picard 2001), that is

$$\mu_z^{(i)} = \mathbb{E}_{q(\mathbf{z}^{(i-1)})} \left[ \mathbf{f}^{(i)}(\mathbf{z}^{(i-1)}; \boldsymbol{\theta}^{(i)}) \right] \tag{2.22}$$

$$\mathbf{v}_z^{(i)} = \mathbb{V}_{q(\mathbf{z}^{(i-1)})} \left[ \mathbf{f}^{(i)}(\mathbf{z}^{(i-1)}; \boldsymbol{\theta}^{(i)}) \right] \tag{2.23}$$

## 2.5 Monte Carlo sampling

This section is a direct copy of section *2.3 Monte Carlo sampling* in Eilertsen 2020. Monte Carlo sampling entails collecting random samples from a distribution. These samples are i.i.d (independent identically distributed), and based on the samples, the mean and the variance of the distribution can be approximated. Monte Carlo sampling is used either when the distribution is intractable, or it is too computationally inefficient to compute it exactly.

Given n independent samples randomly drawn from a distribution. Where each sample is denoted by $x_i$, the mean of the distribution can be approximated as

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.24}$$

this quantity is called the sample mean. The law of large numbers state that given a distribution with expected value, $\mu$ and a finite variance, $\sigma^2$, the sample mean converges almost surely to $\mu$ when $n \to \infty$, which means that it converges with probability equal to one (Evans and J.S.Rosenthal 2004).

The distributions variance can be approximated by

$$s^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{2.25}$$

Generally the error decreases when the amount of samples increase.

## 2.6 MC-dropout

This section is taken from Eilertsen 2020. Gal and Ghahramani 2015 shows that dropout, described in section 2.1.3, Bayesian approximates a deep Gaussian process (Damianou and Lawrence 2013). Which means that predictions made on data that is similar to the

data used during training will have a lower degree of uncertainty associated with them, than predictions made on data which is dissimilar to the data in the training set.

Dropout is applied before every fully connected layer in the network. A fully connected layer is a layer where every unit in the layer is connected to every unit in the previous layer. Dropout is active at test time. At test time the network's prediction is sampled $T$ times for every input. Since dropout is a random process the samples are independent identically distributed, and the samples are therefore Monte Carlo samples, described in 2.5. Thus, the mean and variance of the prediction can be calculated by the following

$$\bar{y} = \frac{1}{T}\sum_{i=1}^{T}\hat{y}_i \tag{2.26}$$

$$v_{model} = \frac{1}{T}\sum_{t=1}^{T}(\hat{y}_i - \bar{y})^2 \tag{2.27}$$

where $\hat{y}_i$ is the network's prediction for each sample.

The optimal dropout rate for estimating the model uncertainty is the same dropout rate used during training. It is also possible to use this method when dropout is not used during training. Then the optimal dropout rate is given by minimizing the negative log-likelihood between predicted, and ground-truth labels (Loquercio, Segu, and Scaramuzza 2020). In practice, the optimal dropout rate for a network trained without dropout is found by grid-search.

Since this method for estimating model uncertainty use Monte Carlo sampling and dropout it is denoted by MC-dropout.

## 2.7 Kalman Filter

The Kalman filter is a recursive state estimator (Kalman 1960). Given a discrete linear system

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}\mathbf{u}_k + w_k \tag{2.28}$$

$$\mathbf{y}_{k+1} = \mathbf{H}\mathbf{x}_{k+1} + v_k \tag{2.29}$$

where $\mathbf{x}_k$ is a vector containing the systems states, $\mathbf{F}$ is the state-transition model, $\mathbf{G}$ is the control input model and $\mathbf{H}$ is the observation model. $\mathbf{u}_k$ is the system's input. Further, $\mathbf{y}_k$ is a measurement of some or all of the system states, and $w_k$ and $v_k$ are white noise process which are zero mean, uncorrelated with known covariance matrices, which are denoted by $\mathbf{Q}_k$ and $\mathbf{R}_k$ respectively. The Kalman filter estimates value of $\mathbf{x}_{k+1}$ which has minimum estimation error covariance based on the noisy measurements of the system, $\mathbf{y}_{k+1}$.

The Kalman filter consists of two steps, a time update step and a measurement update step, also referred to the as the prediction and correction step. This is because the time update step predicts the estimate based on the previous measurement and the measurement step correct this estimate when a new measurement is available. The time update step is given by

$$\begin{aligned} \mathbf{P}_{k+1|k} &= \mathbf{F}\mathbf{P}_{k|k}\mathbf{F}^T + \mathbf{Q}_k \\ \hat{\mathbf{x}}_{k+1|k} &= \mathbf{F}\hat{\mathbf{x}}_{k|k} + \mathbf{B}\mathbf{u}_k \end{aligned} \tag{2.30}$$

where $\hat{\mathbf{x}}_{k+1|k}$ and $\hat{\mathbf{x}}_{k|k}$ are the estimates of $\mathbf{x}_{k+1}$ and $\mathbf{x}_k$ after measurement $\mathbf{y}_k$ is processed. Similarly $\mathbf{P}_{k+1|k}$ and $\mathbf{P}_{k|k}$ are the covariance matrices of the estimation errors of the estimates of $\mathbf{x}_{k+1}$ and $\mathbf{x}_k$ respectively, after measurement $\mathbf{y}_k$ is processed.

The measurement update step is given as

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1|k}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k+1|k}\mathbf{H}^T + \mathbf{R}_{k+1})^{-1} \tag{2.31}$$

$$\hat{\mathbf{x}}_{k+1|k+1} = \hat{\mathbf{x}}_{k+1|k} + \mathbf{K}_{k+1}(\mathbf{y}_{k+1} - \mathbf{H}\hat{\mathbf{x}}_{k+1|k}) \tag{2.32}$$

$$\mathbf{P}_{k+1|k+1} = (I - \mathbf{K}_{k+1}\mathbf{H})\mathbf{P}_{k+1|k}(I - \mathbf{K}_{k+1}\mathbf{H})^T + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^T \tag{2.33}$$

where $\mathbf{K}_{k+1}$ is the Kalman gain.

# 3 Methods

This chapter is organized as the following: first ADF propagation presented in section 2.4 is extended to the case where the network's inputs and the outputs from each layer are assumed dependent, in section 3.1, this method will be denoted by full ADF. Further, expressions for the mean output of a ReLU activation function is given, and the output covariance of the ReLU activation function is derived, in section 3.2.1. Next, in section 3.3 the implementation of full ADF in an ARX network is described together with a method for resetting the estimated variance of the ARX network with a Kalman filter. Then, the implementation of full ADF in a neural network is presented, in section 3.4, and finally in section 3.5 the implementations of both MC-dropout presented in section 2.6 and full ADF will be presented in the context of NARX neural networks.

## 3.1 Full ADF

In this section, ADF, presented in section 2.4, is extended to the case where both the network inputs and output from each network layer are dependent. This extension is done to try to better capture the true aleatoric uncertainty in machine learning models.

The assumptions made in section 2.4 are relaxed. It is now assumed that the output from each network layer is multivariate Gaussian distributed and that the network input is also multivariate Gaussian distributed. Instead of propagating the diagonal of the covariance matrix through the network, the full covariance matrix is propagated, thus this method will be denoted as full ADF.

A linear layer can be expressed as the following

$$\mathbf{z}^{(i+1)} = \mathbf{W}^{(i+1)}\mathbf{z}^{(i)} + \mathbf{b}^{(i+1)} \tag{3.1}$$

where $\mathbf{z}^{(i)}$ is the layers input, and $\mathbf{W}^{(i+1)}$ and $\mathbf{b}^{(i+1)}$ is the layer's weights and biases. When the input is multivariate Gaussian distributed, $\mathbf{z}^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\Sigma}^{(i)})$ the output mean and covariance matrix are given by

$$\boldsymbol{\mu}^{(i+1)} = \mathbf{W}^{(i+1)}\boldsymbol{\mu}^{(i)} + \mathbf{b}^{(i+1)} \tag{3.2a}$$

$$\boldsymbol{\Sigma}^{(i+1)} = \mathbf{W}^{(i+1)}\boldsymbol{\Sigma}^{(i)}(\mathbf{W}^{(i+1)})^T \tag{3.2b}$$

where $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\mu}^{(i+1)}$ is the layer's input and output mean, and $\boldsymbol{\Sigma}^{(i)}$ and $\boldsymbol{\Sigma}^{(i+1)}$ is the layer's input and output covariance matrices.

The output mean of the rectified linear unit function is given in section 3.2. Moreover, the output covariance matrix of the function is derived in the same section when function input is a multivariate Gaussian.

## 3.2 Output mean and covariance of a ReLU activation function

One of the most commonly used activation functions in neural networks is the ReLU activation function (Geoffrey E. Hinton and Neal 1995). The function's definition is given by Equation (2.7) in section 2.1.3.

This section presents expressions for the output mean and variance of a ReLU activation function when the input is an independent Gaussian. Furthermore, the output covariance matrix of the ReLU activation function is derived when the input is a multivariate Gaussian. Moreover, the implementation of the calculation of this mean and covariance is outlined. These expressions are presented and derived because they will be used to implement full ADF, presented in section 3.1, in neural networks, when the ReLU function is used as activation functions.

### 3.2.1 Expressions

Let $X$ be a Gaussian distributed random variable with mean $\mu$ and variance $\sigma^2$, $X \sim \mathcal{N}(\mu, \sigma^2)$, and let $Y$ be the output of the ReLU function, $Y = max\{0, X\}$. The output distribution of a ReLU function is the rectified Gaussian distribution. (Socci, Lee, and Seung 1998). This distribution is a modified Gaussian distribution which is a combination of a discrete distribution that is constant zero and the lower truncated Gaussian distribution on the interval $(0, \infty)$ with a point mass at the origin. Nair and G. Hinton 2010 states that the output mean and variance of a ReLU function is given by

$$\mathbb{E}[Y] = \mu\Phi\left(\frac{\mu}{\sigma}\right) + \sigma\phi\left(\frac{\mu}{\sigma}\right) \tag{3.3}$$

$$\mathbb{V}[Y] = (\mu^2 + \sigma^2)\Phi\left(\frac{\mu}{\sigma}\right) + \mu\sigma\phi\left(\frac{\mu}{\sigma}\right) - \mathbb{E}[Y]^2 \tag{3.4}$$

where $\phi$ and $\Phi$ denotes the probability density function and cumulative distribution function of a standard normal variable, which are given by

$$\begin{aligned} \phi(x) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \\ \Phi(x) &= \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}}\, dt \end{aligned} \tag{3.5}$$

When the ReLU function's input is a random vector, the output will be multivariate rectified Gaussian distributed. Given a normal distributed random vector $\mathbf{X} = \begin{bmatrix} X_1 & X_2 \end{bmatrix}^T$,

where the elements $X_i \sim \mathcal{N}(\mu_{x_i}, \sigma_{x_i}^2)$ and $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_\mathbf{x})$, where

$$\boldsymbol{\mu_x} = \begin{bmatrix} \mu_{x_1} & \mu_{x_2} \end{bmatrix}^T, \quad \boldsymbol{\Sigma}_\mathbf{x} = \begin{bmatrix} \sigma_{x_1}^2 & \rho\sigma_{x_1}\sigma x_2 \\ \rho\sigma_{x_1}\sigma x_2 & \sigma_{x_2}^2 \end{bmatrix}$$

.

Let $\mathbf{Y} = ReLU(\mathbf{X})$, where $\mathbf{Y} = \begin{bmatrix} Y_1 & Y_2 \end{bmatrix}$. The mean and variance of each element in $\mathbf{Y}$ can be found by element-wise applying equations (3.3) and (3.4)

$$\mathbb{E}[Y_i] = \mu_{x_i}\Phi\left(\frac{\mu_{x_i}}{\sigma_{x_i}}\right) + \sigma_{x_i}\phi\left(\frac{\mu_{x_i}}{\sigma_{x_i}}\right) \tag{3.6}$$

$$\mathbb{V}[Y_i] = (\mu_{x_i}^2 + \sigma_{x_i}^2)\Phi\left(\frac{\mu_{x_i}}{\sigma_{x_i}}\right) + \mu_{x_i}\sigma_{x_i}\phi\left(\frac{\mu_{x_i}}{\sigma_{x_i}}\right) - \mathbb{E}[Y_i]^2 \tag{3.7}$$

The output covariance of the ReLU activation function can be found by

$$cov(Y_1, Y_2) = \mathbb{E}[Y_1Y_2] - \mathbb{E}[Y_1]\mathbb{E}[Y_2] \tag{3.8}$$

The product moment, $\mathbb{E}[Y_1Y_2]$, of two truncated variables is given by

$$\mathbb{E}[Y_1Y_2] = \mathbb{E}[X_1X_2 \mid \mathbf{X} > \boldsymbol{\nu}_x]P(\mathbf{X} > \boldsymbol{\nu}_x) + 0P(\mathbf{X} \leq \boldsymbol{\nu}_x) \tag{3.9}$$

where $P$ is the cumulative distribution function of the bivariate normal distribution. $\boldsymbol{\nu}_x$ is a vector containing the variables truncation points, which are zero for both variables.

Since $X_1$ and $X_2$ are Gaussian variables they can be expressed as, $X_1 = \sigma_{x21}U_1 + \mu_{x_1}$ and $X_2 = \sigma_{x_2}U_2 + \mu_{x_2}$ where $U_1$ and $U_2$ are standard normal variables. Thus, equation (3.9) can be rewritten as

$$\begin{aligned} &\mathbb{E}[Y_1Y_2] = \mathbb{E}[(\sigma_{x_1}U_1 + \mu_{x_1})(\sigma_{x_2}U_2 + \mu_{x_2}) \mid \mathbf{X} > \boldsymbol{\nu}_x]P(\mathbf{X} > \boldsymbol{\nu}_x) \\ &= \sigma_{x_1}\sigma_{x_2}\mathbb{E}[U_1U_2 \mid \mathbf{X} > \boldsymbol{\nu}_x]P(\mathbf{X} > \boldsymbol{\nu}_x) + \mu_{x_2}\sigma_{x_1}\mathbb{E}[U_1 \mid \mathbf{X} > \boldsymbol{\nu}_x]P(\mathbf{X} > \boldsymbol{\nu}_x) \\ &\quad + \mu_{x_1}\sigma_{x_2}\mathbb{E}[U_2 \mid \mathbf{X} > \boldsymbol{\nu}_x]P(\mathbf{X} > \boldsymbol{\nu}_x) + \mu_{x_1}\mu_{x_2}P(\mathbf{X} > \boldsymbol{\nu}_x) \\ &= \sigma_{x_1}\sigma_{x_2}\mathbb{E}[U_1U_2 \mid \mathbf{U} > \boldsymbol{\nu}_u]P(\mathbf{U} > \boldsymbol{\nu}_u) + \mu_{x_2}\sigma_{x_1}\mathbb{E}[U_1 \mid \mathbf{U} > \boldsymbol{\nu}_u]P(\mathbf{U} > \boldsymbol{\nu}_u) \\ &\quad + \mu_{x_1}\sigma_{x_2}\mathbb{E}[U_2 \mid \mathbf{U} > \boldsymbol{\nu}_u]P(\mathbf{U} > \boldsymbol{\nu}_u) + \mu_{x_1}\mu_{x_2}P(\mathbf{X} > \boldsymbol{\nu}_x) \end{aligned} \tag{3.10}$$

where $\boldsymbol{\nu}_u$ is a vector containing the truncation points of the standard normal variables. Since the $\mathbf{X}$ is truncated at $X_1 = 0$ and $X_2 = 0$, $\boldsymbol{\nu}_u = \begin{bmatrix} -\frac{\mu_{x_1}}{\sigma_{x_1}} & -\frac{\mu_{x_2}}{\sigma_{x_2}} \end{bmatrix}^T$. According to Rosenbaum 1961 $\mathbb{E}[U_1 \mid \mathbf{U} > \boldsymbol{\nu}_u]P(\mathbf{U} > \boldsymbol{\nu}_u)$, $\mathbb{E}[U_2 \mid \mathbf{U} > \boldsymbol{\nu}_u]P(\mathbf{U} > \boldsymbol{\nu}_u)$ and

$\mathbb{E}[U_1U_2 \mid \mathbf{U} > \boldsymbol{\nu}_u]P(\mathbf{U} > \boldsymbol{\nu}_u)$ are given by

$$\begin{aligned}
\mathbb{E}[U_1 \mid \mathbf{U} > \boldsymbol{\nu}_u]P(\mathbf{U} > \boldsymbol{\nu}_u) &= \phi\left(\frac{\mu_{x_1}}{\sigma_{x_1}}\right)\Phi\left(\frac{\frac{\mu_{x_2}}{\sigma_{x_2}} - \rho\frac{\mu_{x_1}}{\sigma_{x_1}}}{\sqrt{1-\rho^2}}\right) + \\
&\quad \rho\phi\left(\frac{\mu_{x_2}}{\sigma_{x_2}}\right)\Phi\left(\frac{\frac{\mu_{x_1}}{\sigma_{x_1}} - \rho\frac{\mu_{x_2}}{\sigma_{x_2}}}{\sqrt{1-\rho^2}}\right) \\
\mathbb{E}[U_2 \mid \mathbf{U} > \boldsymbol{\nu}_u]P(\mathbf{U} > \boldsymbol{\nu}_u) &= \rho\phi\left(\frac{\mu_{x_1}}{\sigma_{x_1}}\right)\Phi\left(\frac{\frac{\mu_{x_2}}{\sigma_{x_2}} - \rho\frac{\mu_{x_1}}{\sigma_{x_1}}}{\sqrt{1-\rho^2}}\right) + \\
&\quad \phi\left(\frac{\mu_{x_2}}{\sigma_{x_2}}\right)\Phi\left(\frac{\frac{\mu_{x_1}}{\sigma_{x_1}} - \rho\frac{\mu_{x_2}}{\sigma_{x_2}}}{\sqrt{1-\rho^2}}\right) \\
\mathbb{E}[U_1U_2 \mid \mathbf{U} > \boldsymbol{\nu}_u]P(\mathbf{U} > \boldsymbol{\nu}_u) &= \rho P(\mathbf{U} > \boldsymbol{\nu}_u) - \\
&\quad \rho\frac{\mu_{x_2}}{\sigma_{x_2}}\phi\left(\frac{\mu_{x_2}}{\sigma_{x_2}}\right)\Phi\left(\frac{\frac{\mu_{x_2}}{\sigma_{x_2}} - \rho\frac{\mu_{x_1}}{\sigma_{x_1}}}{\sqrt{1-\rho^2}}\right) - \\
&\quad \rho\frac{\mu_{x_2}}{\sigma_{x_2}}\phi\left(\frac{\mu_{x_2}}{\sigma_{x_2}}\right)\Phi\left(\frac{\frac{\mu_{x_1}}{\sigma_{x_1}} - \rho\frac{\mu_{x_2}}{\sigma_{x_2}}}{\sqrt{1-\rho^2}}\right) + \\
&\quad \frac{\sqrt{1-\rho^2}}{\sqrt{2\pi}}\phi\left(\frac{\sqrt{(\frac{\mu_{x_1}}{\sigma_{x_1}})^2 - 2\rho\frac{\mu_{x_1}}{\sigma_{x_1}}\frac{\mu_{x_2}}{\sigma_{x_2}} + (\frac{\mu_{x_2}}{\sigma_{x_2}})^2}}{\sqrt{1-\rho^2}}\right)
\end{aligned} \tag{3.11}$$

where $\rho$ is the correlation between $X_1$ and $X_2$. Combining equations (3.10) and (3.11) yields

$$\begin{aligned}
\mathbb{E}[Y_1Y_2] &= P(\mathbf{X} > \boldsymbol{\nu}_x)(\mu_{x_1}\mu_{x_2} + \sigma_{x_1}\sigma_{x_2}\rho) + \\
&\quad \mu_{x_1}\sigma_{x_2}\phi\left(\frac{\mu_{x_2}}{\sigma_{x_2}}\right)\Phi\left(\frac{\frac{\mu_{x_1}}{\sigma_{x_1}} - \rho\frac{\mu_{x_2}}{\sigma_{x_2}}}{\sqrt{1-\rho^2}}\right) + \\
&\quad \mu_{x_2}\sigma_{x_1}\phi\left(\frac{\mu_{x_1}}{\sigma_{x_1}}\right)\Phi\left(\frac{\frac{\mu_{x_2}}{\sigma_{x_2}} - \rho\frac{\mu_{x_1}}{\sigma_{x_1}}}{\sqrt{1-\rho^2}}\right) + \\
&\quad \sigma_{x_1}\sigma_{x_2}\frac{\sqrt{1-\rho^2}}{\sqrt{2\pi}}\phi\left(\sqrt{\frac{\left(\frac{\mu_{x_1}}{\sigma_{x_1}}\right)^2 - \frac{2\rho\mu_{x_1}\mu_{x_2}}{\sigma_{x_1}\sigma_{x_2}} + \left(\frac{\mu_{x_2}}{\sigma_{x_2}}\right)^2}{1-\rho^2}}\right)
\end{aligned} \tag{3.12}$$

Inserting equation (3.12) into equation (3.8) gives the following expression for the output

covariance of a ReLU activation function

$$
\begin{aligned}
cov(Y_1, Y_2) = P(\mathbf{X} > \boldsymbol{\nu}_x)(\mu_{x_1}\mu_{x_2} + \sigma_{x_1}\sigma_{x_2}\rho) + \\
\mu_{x_1}\sigma_{x_2}\phi\left(\frac{\mu_{x_2}}{\sigma_{x_2}}\right)\Phi\left(\frac{\frac{\mu_{x_1}}{\sigma_{x_1}} - \rho\frac{\mu_{x_2}}{\sigma_{x_2}}}{\sqrt{1-\rho^2}}\right) + \\
\mu_{x_2}\sigma_{x_1}\phi\left(\frac{\mu_{x_1}}{\sigma_{x_1}}\right)\Phi\left(\frac{\frac{\mu_{x_2}}{\sigma_{x_2}} - \rho\frac{\mu_{x_1}}{\sigma_{x_1}}}{\sqrt{1-\rho^2}}\right) + \\
\sigma_{x_1}\sigma_{x_2}\frac{\sqrt{1-\rho^2}}{\sqrt{2\pi}}\phi\left(\sqrt{\frac{\left(\frac{\mu_{x_1}}{\sigma_{x_1}}\right)^2 - \frac{2\rho\mu_{x_1}\mu_{x_2}}{\sigma_{x_1}\sigma_{x_2}} + \left(\frac{\mu_{x_2}}{\sigma_{x_2}}\right)^2}{1-\rho^2}}\right) - \\
\mathbb{E}[Y_1]\mathbb{E}[Y_2]
\end{aligned}
\tag{3.13}
$$

### 3.2.2 Implementation

This section describes the implementation of the calculation of the output mean and covariance of a ReLU unit. In this section $\circ$ denotes the element-wise product, all exponents, divisions and square roots are preformed element-wise. Moreover, $\phi$ and $\Phi$ denotes the probability density function and cumulative distribution function respectively of a standard normal variable, both functions preforms element-wise operations. It has been implemented in Python with the Pytorch framework (Van Rossum and Drake 2009)(Paszke et al. 2019).

Given a Gaussian random vector $\mathbf{X} = \begin{bmatrix} X_1, \cdots, X_n \end{bmatrix}$ where $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$, where

$$
\boldsymbol{\mu}_x = \begin{bmatrix} \mu_{x_1}, \cdots, \mu_{x_n} \end{bmatrix}, \quad \boldsymbol{\Sigma}_x = \begin{bmatrix} \sigma_{x_1}^2 & \cdots & \rho_{1n}\sigma_{x_1}\sigma x_n \\ \vdots & \ddots & \vdots \\ \rho_{1n}\sigma_{x_1}\sigma x_n & \cdots & \sigma_{x_n}^2 \end{bmatrix}
$$

Let $\boldsymbol{\sigma}_x$ denote element-wise square root of the diagonal of $\boldsymbol{\Sigma}_x$, $\boldsymbol{\sigma}_x = \sqrt{\text{diag}(\boldsymbol{\Sigma}_x)}$ and let $\mathbf{Y} = Relu(\mathbf{X})$. The mean and the diagonal elements of the covariance matrix of $\mathbf{Y}$ can be found with equations (3.3) and (3.3), but some extra precautions have to be taken when one or more of the elements of $\boldsymbol{\sigma}_x$ is equal to zero. Since $\sigma_{x_i} = \sqrt{\sigma_{x_i}^2}$

$$
\lim_{\sigma_{x_i} \to 0} \frac{\mu_{x_i}}{\sigma_{x_i}} = \begin{cases} 0, & \text{if } \mu_{x_i} = 0 \\ \infty \times \text{sgn}(\mu_{x_i}), & \text{if } \mu_{x_i} \neq 0 \end{cases} \tag{3.14}
$$

Therefore, the calculation of the mean of a ReLU unit can be implemented as the following

$$
\mathbb{E}[\mathbf{Y}_i] = \begin{cases} \mu_{x_i}\phi\left(\frac{\mu_{x_i}}{\sigma_{x_i}}\right) + \sigma_{x_i}\Phi\left(\frac{\mu_{x_i}}{\sigma_{x_i}}\right), & \text{if } \sigma_{x_i} \neq 0 \\ 0, & \text{if } \sigma_{x_i} = 0 \text{ and } \mu_{x_i} = 0 \\ \mu_{x_i}\Phi\left(\infty \times \text{sgn}(\mu_{x_i})\right), & \text{if } \sigma_{x_i} = 0 \text{ and } \mu_{x_i} \neq 0 \end{cases} \tag{3.15}
$$

Similar precautions has to be taken regarding the diagonal elements of the covariance matrix of $\mathbf{Y}$. Thus, the output variance of $\mathbf{Y}$ can be found by

$$\mathbb{V}[\mathbf{Y}_i] = \begin{cases} (\mu_{x_i}^2 + \sigma_{x_i}^2)\Phi\left(\frac{\mu_{x_i}}{\sigma_{x_i}}\right) + \mu_{x_i}\sigma_{x_i}\phi\left(\frac{\mu_{x_i}}{\sigma_{x_i}}\right) - \mathbb{E}[Y_i]^2, & \text{if } \sigma_{x_i} \neq 0 \\ 0, & \text{if } \sigma_{x_i} = 0 \text{ and } \mu_{x_i} = 0 \\ \mu_{x_i}^2 \Phi\left(\text{sgn}(\mu_{x_i}) \times \infty\right) - \mathbb{E}[Y_i]^2, & \text{if } \sigma_{x_i} = 0 \text{ and } \mu_{x_i} \neq 0 \end{cases} \tag{3.16}$$

To easily calculate the off diagonal elements of the covariance matrix, defined by equation (3.13), some matrices has to be defined. Let the matrix $\boldsymbol{\rho}$ and the $n \times n$ matrix $\mathbf{A}$ be defined as the following

$$\boldsymbol{\rho} = \frac{\boldsymbol{\Sigma}_x}{\boldsymbol{\sigma}_x(\boldsymbol{\sigma}_x)^T} = \begin{bmatrix} \sigma_{x_1} & \cdots & \rho_{1n} \\ \vdots & \ddots & \vdots \\ \rho_{1n} & \cdots & \sigma_{x_n} \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} — & \frac{\boldsymbol{\mu_x}}{\boldsymbol{\sigma_x}} & — \\ & \vdots & \\ — & \frac{\boldsymbol{\mu_x}}{\boldsymbol{\sigma_x}} & — \end{bmatrix}_{n \times n}$$

and the let matrix $\mathbf{B}$ be defined as

$$\mathbf{B} = \boldsymbol{\mu}_x(\boldsymbol{\sigma}_x)^T \circ \phi(\mathbf{A}) \circ \Phi\left(\frac{\mathbf{A}^T - \boldsymbol{\rho} \circ \mathbf{A}}{\sqrt{\mathbf{1} - \boldsymbol{\rho}^2}}\right) \tag{3.17}$$

Let $\mathbf{P}$ be the following matrix

$$\mathbf{P} = \begin{bmatrix} 0 & p_{12} & \cdots & p_{1n} \\ p_{12} & \ddots & \cdots & p_{2n} \\ \vdots & & \ddots & p_{n-1n} \\ p_{1n} & \cdots & p_{n-1n} & 0 \end{bmatrix}$$

where $p_{ij}$ denotes $P([X_i, X_j] > [0, 0])$, where $P$ is the bivariate cumulative distribution function. These probabilities are obtained with the Scipy framework (Virtanen et al. 2020), as the Pytorch framework does not support this functionality.

The off diagonal elements of the output covariance matrix, described by equation (3.13), can then be calculated with the following expression

$$\begin{aligned} \boldsymbol{\Sigma}_y = \mathbf{P} \circ (\boldsymbol{\mu}_x(\boldsymbol{\mu}_x)^T - \boldsymbol{\Sigma}_x) + \mathbf{B} + \mathbf{B}^T + \\ \boldsymbol{\sigma}_x(\boldsymbol{\sigma}_x)^T \circ \frac{\sqrt{\mathbf{1} - \boldsymbol{\rho}^2}}{\sqrt{2\pi}} \circ \phi\left(\sqrt{\frac{(\mathbf{A}^T)^2 - 2\boldsymbol{\rho} \circ \left(\frac{\boldsymbol{\mu_x}}{\boldsymbol{\sigma_x}}\right)^T \left(\frac{\boldsymbol{\mu_x}}{\boldsymbol{\sigma_x}}\right) + \mathbf{A}^2}{\mathbf{1} - \boldsymbol{\rho}^2}}\right) - (\mathbb{E}[Y])^T \mathbb{E}[Y] \end{aligned} \tag{3.18}$$

As with the calculation of the mean and the diagonal of the covariance matrix there are some special considerations which has to be taken into account when computing the off diagonal elements of the covariance matrix. The first one is when two or more of

the input variables are perfectly correlated or perfectly negatively correlated. In other words, when $\rho_{ij}$ is equal to one or negative one. The second to last term in equation (3.18) will be zero, and the $\mathbf{B}$ matrix in the equation will change to the following if it is expressed element-wise

$$b_{ij} = \begin{cases} \mu_{x_i}\sigma_{x_j}\phi\left(\frac{\mu_{x_j}}{\sigma_{x_j}}\right)\Phi\left(0\right), & \text{if } |\rho_{ij}| = 1 \ and \ \frac{\mu_{x_i}}{\sigma_{x_i}} - \rho_{ij}\frac{\mu_{x_j}}{\sigma_{x_j}} = 0 \\ \mu_{x_i}\sigma_{x_j}\phi\left(\frac{\mu_{x_j}}{\sigma_{x_j}}\right)\Phi\left(\infty \times \text{sgn}(\frac{\mu_{x_i}}{\sigma_{x_i}} - \rho_{ij}\frac{\mu_{x_j}}{\sigma_{x_j}})\right), & \text{if } |\rho_{ij}| = 1 \end{cases} \tag{3.19}$$

Furthermore, when the calculation of the covariance is done in a neural network some numerical rounding issues may arise, leading to the input covariance matrix not being positive semi-definite. And thus, it is not possible to calculate the entries in the $\mathbf{P}$ matrix. To circumvent this problem a very small number is added to the diagonal of covariance matrix of the input. In the implementation used in this thesis the following is done, if the covariance matrix has a negative eigenvalue, the absolute value of the eigenvalue is added to the covariance matrix's diagonal.

## 3.3 Estimating aleatoric uncertainty in an ARX network

In this section, a method for estimating aleatoric uncertainty with full ADF, presented in 3.1, in an ARX network, described in section 2.2.1 which approximates a linear process is outlined. Furthermore, a mechanism for resetting the estimated uncertainty is presented when measurements from the process are available. This is done with a Kalman filter, described in section 2.7, where the Kalman filter's state transition model and control input model are given by the ARX network's model parameters.

The network which implements full ADF will be denoted by full ADF ARX network. The network will predict the mean value of the process and estimate the output variance when the process is affected by a disturbance with known mean and variance.

The section is organized as follows: first the ARX model which is approximated by the network is presented and the ARX network is described. Thereafter training of the network is outlined. Finally the resetting of the estimated variance with a Kalman filter is presented.

### 3.3.1 Data

The process which the ARX network is approximating is given by the following equation

$$y_{k+1} = y_k - 0.5y_{k-1} + 3u_k - 2u_{k-1} + w_{k+1} \tag{3.20}$$

This is an ARX model, which is described in section 2.2, where $y_k$ is the process output at time step $k$ and $u_k$ is the process input at time step $k$. Both the process input and output are scalar valued. $w_{k+1}$ is a disturbance which is Gaussian distributed with zero

mean and variance of 1, $w_{k+1} \sim \mathcal{N}(0, 1)$. The disturbance is uncorrelated in time. The ARX model can be written on state space form as

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{p}_{k+1} \tag{3.21}$$

$$y_{k+1} = \mathbf{C}\mathbf{x}_{k+1} \tag{3.22}$$

where

$$\mathbf{x}_k = \begin{bmatrix} u_{k-1} \\ y_{k-1} \\ y_k \end{bmatrix} \quad \mathbf{p}_{k+1} = \begin{bmatrix} u_k \\ w_{k+1} \end{bmatrix} \mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ -2 & -0.5 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 3 & 1 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

The mean of $\mathbf{x}_{k+1}$ is given by

$$\boldsymbol{\mu}^x_{k+1} = \mathbb{E}[\mathbf{x}_{k+1}] = \mathbf{A}\boldsymbol{\mu}^x_k + \mathbf{B}\boldsymbol{\mu}^p_{k+1} \tag{3.23}$$

where $\boldsymbol{\mu}^p_{k+1} = \mathbb{E}[\mathbf{p}_{k+1}]$ and the covariance matrix of $\mathbf{x}_{k+1}$ is given by

$$\boldsymbol{\Sigma}^x_{k+1} = \mathbb{E}[(\mathbf{x}_{k+1} - \boldsymbol{\mu}^x_{k+1})(\mathbf{x}_{k+1} - \boldsymbol{\mu}^x_{k+1})^T] = \mathbf{A}\boldsymbol{\Sigma}^x_k(\mathbf{A})^T + \mathbf{B}\boldsymbol{\Sigma}^p_{k+1}(\mathbf{B})^T \tag{3.24}$$

where $\boldsymbol{\Sigma}^p_{k+1} = \mathbb{E}[(\mathbf{p}_{k+1} - \boldsymbol{\mu}^p_{k+1})(\mathbf{p}_{k+1} - \boldsymbol{\mu}^p_{k+1})^T]$, assuming that the ARX model's inputs are deterministic.

The training set has been generated by creating several different time varying input signals, and simulating the ARX model with these input signals.

### 3.3.2 Network

The ARX network which is used to approximate the ARX model is illustrated in Figure 3.1. The network has five inputs, one linear layer, and two outputs. The network predicts both $\hat{y}_{k+1}$ and $\hat{y}_k$, although $\hat{y}_k$ is known because it is used to estimate the covariance between the two predictions at each time step, $\Sigma_{\hat{y}_{k-1}\hat{y}_k}$ which is used to estimate the variance in the next time step, when the network is a full ADF ARX network.

The network is converted to a full ADF ARX network by replacing the linear layer with a linear layer which propagates means and covariances, which is given by Equation (3.2). The full ADF ARX network's input mean vector with corresponding input covariance vector is as follows

$$\begin{aligned} \mathbf{x}^n_k &= \begin{bmatrix} u_k & u_{k-1} & \mu_{\hat{y}_{k-1}} & \mu_{\hat{y}_k} & \mu_{w_{k+1}} \end{bmatrix} \\ \boldsymbol{\Sigma}^n_k &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{\hat{y}_{k-1}} & \Sigma_{\hat{y}_{k-1}\hat{y}_k} & 0 \\ 0 & 0 & \Sigma_{\hat{y}_{k-1}\hat{y}_k} & \sigma^2_{\hat{y}_k} & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{w_{k+1}} \end{bmatrix} \end{aligned} \tag{3.25}$$

where $\sigma^2_{\hat{y}_{k-1}}$ is the estimated variance of $\hat{y}_{k-1}$, $\sigma^2_{\hat{y}_k}$ is the estimated variance of $\hat{y}_k$, $\sigma^2_{w_{k+1}}$ is the variance of $w_{k+1}$ and $\Sigma_{\hat{y}_{k-1}\hat{y}_k}$ is the estimated covariance between $\hat{y}_{k+1}$ and $\hat{y}_k$. $w_{k+1}$ is equal to zero for all predictions, and $\sigma^2_{w_{k+1}}$ is equal to one for all predictions.

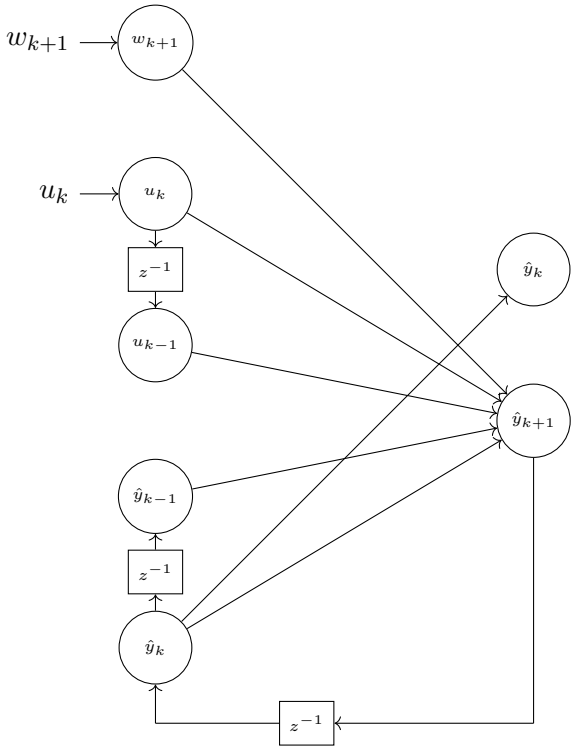



Figure 3.1: Illustration of ARX neural network used to approximate the ARX model described by Equation (3.20). The network has five inputs, one linear layer, and two outputs.

At each time step the full ADF ARX network predicts the mean of $\hat{y}_{k+1}$ and $\hat{y}_k$, $\mu_{\hat{y}_{k+1}}$ and $\mu_{\hat{y}_k}$ and estimates their covariance matrix. This can be expressed as

$$\hat{\mathbf{y}}_{k+1} = \begin{bmatrix} \mu_{\hat{y}_k} \\ \mu_{\hat{y}_{k+1}} \end{bmatrix} \quad \Sigma_{\hat{\mathbf{y}}_{k+1}} = \begin{bmatrix} \sigma^2_{\hat{y}_k} & \Sigma_{\hat{y}_{k-1}\hat{y}_k} \\ \Sigma_{\hat{y}_{k-1}\hat{y}_k} & \sigma^2_{\hat{y}_{k+1}} \end{bmatrix}$$

At each time step the input vector and input covariance matrix are updated with values obtained in the previous time step.

### 3.3.3 Training

The network is trained as a linear feedforward network. Since the relation between the network's input and $y_k$ is known, it does not have to be learned. Furthermore, since the network input $w_{k+1}$ is additive white noise affecting the prediction, its relation with the prediction cannot be learned. However, since the mean and variance of the noise known and the goal of full ADF propagation is to predict the mean and estimate the variance, it does not need to be learned. Therefore, the network only needs to learn the relation between the inputs $u_k$, $u_{k-1}$, $y_k$ and $y_{k-1}$ and the output $y_{k+1}$. This relation can be expressed as

$$\hat{y}_{k+1} = w_1 u_{k-1} + w_2 u_k + w_3 y_k + w_4 y_{k-1} \tag{3.26}$$

Then the whole feedforward network can be described by

$$\begin{bmatrix} \hat{y}_k \\ \hat{y}_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ w_1 & w_2 & w_3 & w_4 & 1 \end{bmatrix} \begin{bmatrix} u_{k-1} \\ u_k \\ y_k \\ y_{k-1} \\ w_{k+1} \end{bmatrix} \tag{3.27}$$

Where the model parameters $w_1$, $w_2$, $w_3$ and $w_4$ are found with Equation (2.4) given in section 2.1.2.

### 3.3.4 Evaluation

To evaluate both the full ADF ARX network's performance and variance estimate, the MSE given by Equation (2.9) in section 2.1.3 will be used. The mean estimate will be compared to the output mean of the ARX model described by Equation (3.23), and variance estimate will be compared with the output variance of the ARX model given by Equation (3.24).

### 3.3.5 Kalman Filter

A Kalman filter presented in section 2.7 will be used to update the full ADF ARX network's input covariance when a measurement from the ARX model is available. When

measurements are unavailable the Kalman filter will be updated with the output mean and covariance from the full ADF ARX network and the ARX model's input. The Kalman filter can be expressed as

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}\mathbf{u}_k + w_{k+1} \tag{3.28}$$

$$y_{k+1} = \mathbf{H}\mathbf{x}_{k+1} + v_{k+1} \tag{3.29}$$

where

$$\mathbf{x}_k = \begin{bmatrix} y_{k-1} \\ y_k \end{bmatrix} \quad \mathbf{u}_k = \begin{bmatrix} u_{k-1} \\ u_k \end{bmatrix}$$

and

$$\mathbf{F} = \begin{bmatrix} 0 & 1 \\ w_4 & w_3 \end{bmatrix} \quad \mathbf{G} = \begin{bmatrix} 0 & 0 \\ w_1 & w_2 \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

where $w_1$, $w_2$, $w_3$ and $w_4$ are the ARX networks model parameters. $w_{k+1}$ is the same disturbance as in Equation (3.20). Therefore, the $\mathbf{Q}$ matrix is given as

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$v_{k+1}$ is noise affecting the measurements of the ARX model. Since the filter is either updated with measurements from the ARX model which are noise free or from the mean predictions made by the full ADF ARX network which have an associated estimated variance, $R_{k+1} = 0$ if a measurement from the ARX model is available and $R_{k+1} = \sigma^2_{\hat{y}_{k+1}}$ if the measurement is not available.When a measurement from the ARX model

**Algorithm 1:** Full ADF ARX network with Kalman filter

Initialize full ADF ARX network's mean input vector, $\boldsymbol{\mu}^n_{x_0}$, and covariance input vector $\boldsymbol{\Sigma}^n_{x_0}$
Initialize the Kalman filter's priori estimate covariance matrix, $\mathbf{P}_0$, and initial state vector, $\mathbf{x}_0$
**for** *k = 1, 2, . . . , N* **do**
    Make prediction and variance estimate with full ADF ARX network
    **if** *Measurement is available* **then**
        Update Kalman filter with the measurement and $R_{k+1} = 0$
        Update network's output covariance matrix with covariance matrix from Kalman filter, $\Sigma_{\hat{\mathbf{y}}_{k+1}} = \mathbf{P}_{k+1|k+1}$
    **else**
        Update Kalman filter with prediction, $\mu_{\hat{y}_k+1}$, and $R_{k+1} = \sigma^2_{\hat{y}_k+1}$
    **end**
    Update network's input mean vector and input covariance matrix.
**end**

is available the output covariance matrix of the full ADF ARX network is replaced by the Kalman filter's covariance matrix of the estimation errors, after the Kalman filter's correction step is preformed. The whole procedure is summarized in Algorithm 1.

## 3.4 Estimating aleatoric uncertainty in a nonlinear network with full ADF

In this section, a method for estimating aleatoric uncertainty with full ADF, presented in 3.1, in neural network, described in section 2.2.1 is outlined. Two networks are trained to approximate the same system. One of the networks is trained on normalized data and the other network is trained on data which is not normalized.

This section is organized as the following: first the system which the networks are approximating is presented together with how the training set is created. Then, the implementations and training of the networks are described. Followed by a description of how the networks will be evaluated, both in terms of mean predictive performance and variance estimate performance.

### 3.4.1 Data

The uncertainty estimation method will be be tested on the following system

$$y = x^2 \tag{3.30}$$

where $x$ is the system's input and $y$ is the system's output. The output variance of this system when the input is Gaussian distributed, $X \sim \mathcal{N}(\mu_x, \sigma_x^2)$, is given by

$$\sigma_y^2 = 2\sigma_x^4 + 4\mu_x^2\sigma_x^2 \tag{3.31}$$

The training set has been created by applying uniformly distributed inputs to the system on the interval $[0, 20]$, and corrupting the outputs of the system with Gaussian white noise with a mean equal to zero and variance equal to $0.1^2$. Two training set where created, one were the inputs and the outputs are normalized between 0 and 1, and one which is not normalized.

### 3.4.2 Networks

Two networks were trained to evaluate the aleatoric uncertainty estimate obtained from full ADF. One was trained on the training set that is not normalized, and one trained on the normalized training set. Both networks were trained with mini-batch gradient descent with the MSE-loss function, presented in section 2.1.3. Moreover, L2-regularization described in section 2.1.3 was applied to prevent overfitting. The ADAM-optimizer(Kingma and Ba 2017) with learning rate $1 \times 10^{-3}$ was used to train both networks. Furthermore, the networks were trained as deterministic networks and converted to full ADF networks after the training process was finished. Both networks consists of one hidden layers with ten units, where the hidden layer has ReLU activation functions. The mean and the covariance matrix of the linear layers in a full ADF network are given by Equation (3.2a) and Equation (3.2b) respectively. The output mean and

output covariance of the ReLU activation functions are given in section 3.2.1. The full ADF network's input vector contains the input mean and input variance, and can be expressed as

$$\mathbf{x}_i = (\mu_{x_i}, \sigma^2_{x_i})$$

for each input i. The output vector of the system is expressed as

$$\hat{\mathbf{y}}_i = (\mu_{\hat{y}_i}, \hat{\sigma}^2_{\hat{y}_i})$$

where $\mu_{\hat{y}_i}$ is the mean of prediction $i$ and $\hat{\sigma}^2_{\hat{y}_i}$ is the estimated variance of the prediction.

### 3.4.3 Evaluation

The MSE, described by Equation (2.9) in section 2.1.3, is used to evaluate both the networks predictive performance and uncertainty estimates. The MSE of the uncertainty estimates will be calculated against the output variance of the system, described by equation (3.31). Furthermore, will the networks estimated variances be compared to the output sample variance of the networks with the purpose to evaluate how accurately full ADF estimates the true output variance of the network. The sample output variance of the networks will be obtained with Monte Carlo sampling described in section 2.5. The i.i.d samples of the networks are acquired by drawing independent samples from a Gaussian distribution which have the same mean and variance as the input given to the full ADF networks for each prediction. The sample output variance is obtained on the deterministic networks. $10^8$ samples are used to obtain each sample output variance. The output sample variance will be denoted by $S^2_{\hat{y}}$.

## 3.5 Uncertainty estimation in NARX neural networks

In this section, two methods for estimating uncertainty in a NARX neural network is presented. MC-dropout for estimating epistemic uncertainty, presented in section 2.6 and full ADF for estimating aleatoric uncertainty, described in section 3.1 is presented in the context of a NARX neural network, detailed in section 2.2.2. The network which implements MC-dropout will be denoted by MC-dropout The network which implements full ADF will be denoted by full ADF network.

The aim of the full ADF network is to estimate the aleatoric variance associated with each prediction which is caused by the variance associated with the network's initial input. Unlike in full ADF ARX networks presented in section 3.3, where it is trivial to find the covariance between subsequent predictions, since the network is linear, this is not the case in full ADF NARX neural networks. Therefore, it will be assumed that the network predictions are independent. It is also assumed that the initial mean value of the inputs are known and that the variance associated with these inputs is the measurement noise of the process output.

This section is organized as follows, first the process which the estimation methods are tested on is introduced and then a description of how the data sets are created. Further, the implementation, training and evaluation of the networks is presented.

### 3.5.1 Data

This section is a rewritten version of section *3.1 Data* in Eilertsen 2020. In this section, the system which the uncertainty estimation methods are tested on is presented. Further, how the training set is created is described.

#### Model

The data for testing the different methods for computing uncertainty was obtained from simulations of a continuous stirred-tank reactor (CSTR) process (Seborg, Edgar, and Mellichamp 1989). This is a chemical process where a reactant A is converted to a product B through a chemical reaction. The CSTR model is described by

$$\dot{C_A}(t) = \frac{q_0}{V}(C_f - C_A(t)) - k_0 e^{-\frac{-E}{RT(t)}} C_A(t) \tag{3.32}$$

$$\dot{T}(t) = \frac{q_0}{V}(T_f - T(t)) - \frac{\Delta H_r}{\rho C_p} e^{\frac{-E}{RT(t)}} C_A(t) + \frac{UA}{V\rho C_p}(T_c(t) - T(t)) \tag{3.33}$$

$$\dot{T_c}(t) = \frac{T_r(t) - T_c(t)}{\tau} \tag{3.34}$$

where $C_A$ $[mol/l]$ is the concentration of reactant $A$ in the tank, $T$ $[K]$ is the temperature in the tank and $T_c$ $[K]$ is the coolant temperature. $T_r$ $[K]$ is the reference temperature for $T_c$. The model parameters are given in Table 3.1, the model parameters are the same as used by Manzano et al. 2019. $T_r$ is the process input and $C_A$ is the output. And accordingly $T_r$ are the $u$'s in the NARX neural network and $C_A$ are the $y$'s. $T_r$ has been constrained to the interval $335\,\mathrm{K} \leq u \leq 370\,\mathrm{K}$, which corresponds to $0.2\,\mathrm{mol/l} \leq y \leq 0.81\,\mathrm{mol/l}$. Moving forward these intervals will be referred to as the state space.

#### Datasets

The training data set have been generated by applying a sweeping chirp signal as the coolant temperature reference. A sweeping chirp signal is a signal where the frequency varies with time. This signal is persistently exciting, which makes it possible to perform system identification. When creating the data sets, it is assumed that the process is steady-state before the time-varying input signal is applied. A constant input signal, known to bring the process to steady state, was applied at the beginning of the simulations to achieve this. The process was simulated with forward Euler with a time step of 0.5 min. The inputs where normalized between 0 and 1 to ease the training process. Two training sets were created. The first training set, visualized in Figure 3.2, contains a subset of the state space, where $u \in (345\,\mathrm{K}, 370\,\mathrm{K})$ which corresponds

| Parameter | Value | Definition |
|---|---|---|
| $q_0$ | $10\,\mathrm{l/min}$ | Reactive input flow |
| V | $150\,\mathrm{l}$ | Liquid volume in the tank |
| $k_0$ | $6 \times 10^{10}\,\mathrm{l/min}$ | Frequency constant |
| $\frac{E}{R}$ | $9750\,\mathrm{K}$ | Arrhenius constant |
| $-\Delta H_r$ | $10^4\,\mathrm{J/mol}$ | Reaction enthalpy |
| UA | $7 \times 10^4\,\mathrm{J/(min\,K)}$ | Heat transfer coefficient |
| $\rho$ | $1100\,\mathrm{g/l}$ | Density |
| $C_p$ | $0.3\,\mathrm{J/(g\,K)}$ | Specific heat |
| $\tau$ | $1.5\,\mathrm{min}$ | Time constant |
| $C_f$ | $1\,\mathrm{mol/l}$ | $C_a$ in input flow |
| $T_f$ | $370\,\mathrm{K}$ | Input flow temperature |

Table 3.1: CSTR-process model parameters

to $y \in (0.2\,\mathrm{mol/l}, 0.6466\,\mathrm{mol/l})$, moving forward this subset will be refereed to as the training space. The inputs and outputs in the state space which are not represented in the training space are $u \in (335\,\mathrm{K},\ 345\,\mathrm{K})$ and $y \in (0.6466\,\mathrm{mol/l},\ 0.81\,\mathrm{mol/l})$. Note that not every part of the training space is represented equally in the training set. The subset $[354.5\,\mathrm{K}, 360.5\,\mathrm{K}]$ which corresponds to $y \in [0.3596\,\mathrm{mol/l}, 0.4670\,\mathrm{mol/l}]$ is most represented in the training set.

The second training set contains the whole state space with added white Gaussian noise to the processes output signal, y. The noise has zero mean and a standard deviation of $0.015\,25\,\mathrm{mol/l}$, which corresponds to a variance of 2.5 % of the magnitude of the space spanned by $y$. The process input is not affected by noise.

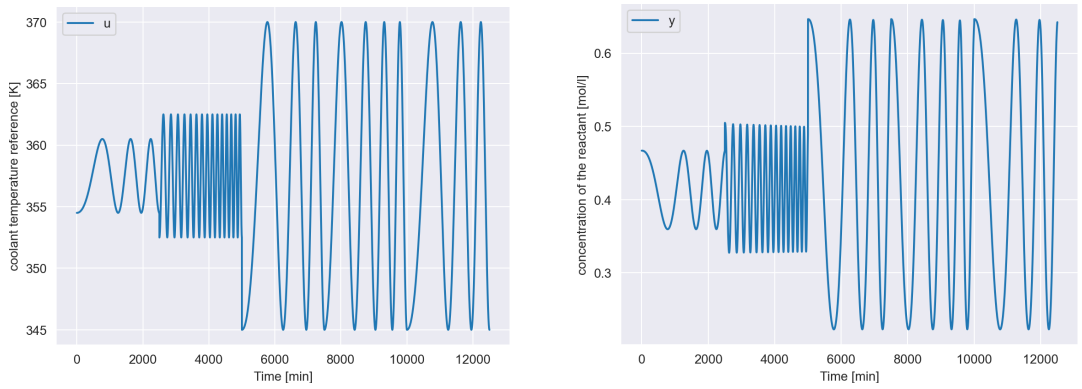
The CSTR-process dynamics are assumed unknown to the networks. The networks only sees the process inputs and outputs.

### 3.5.2 Implementation

In this section, the implementation, training and evaluation of the networks are described. The neural networks have been implemented in Python with the Pytorch framework (Van Rossum and Drake 2009)(Paszke et al. 2019).

#### NARX neural network

This section is a rewritten version of section *NARX neural network* in (Eilertsen 2020). The NARX neural network, presented in 2.2.2, is implemented as a fully connected feedforward network, where all the layers are linear. Every unit in the hidden layers have ReLU activation functions. The number of hidden layers and the number of units




(a) CSTR-process input

(b) CSTR-process output

Figure 3.2: Training set which contains a subset of the state space, which is denoted by the training space. The training space consists of inputs and outputs in the intervals, $u \in (345\,\mathrm{K}, 370\,\mathrm{K})$, $y \in (0.2\,\mathrm{mol/l}, 0.6466\,\mathrm{mol/l})$. Note that there are more training examples in the middle of these intervals and that all the examples are time varying. The part of the state space that is not represented in the training set are inputs in the interval (335 K, 345 K) which corresponds to outputs in the interval (0.65 mol/l, 0.81 mol/l).

in each layer can be chosen. In the implementation, the number of units in each hidden layer is the same.

The network's input vector for a time step $k$ can be expressed as

$$\mathbf{x}_k = \begin{bmatrix} u_k & \cdots & u_{k-m_u} & \hat{y}_{k-m_y} & \cdots & \hat{y}_k \end{bmatrix} \tag{3.35}$$

where $m_u$ and $m_y$ describe how many previous process inputs and outputs respectively are used to obtain a prediction. This input vector is forward passed through the network to obtain the output $\hat{y}_{k+1}$, the network's prediction. This output is then added to the input vector at the next time step $k+1$, while concurrently $\hat{y}_{k-m_y}$ and $u_{k-m_y}$ are removed, and the process input $u_k$ is added to the input vector. Thus, the input vector at time $u_{k+1}$ is

$$\mathbf{x}_{k+1} = \begin{bmatrix} u_{k+1} & \cdots & u_{k-m_u+1} & \hat{y}_{k-m_y+1} & \cdots & \hat{y}_{k+1} \end{bmatrix} \tag{3.36}$$

where $u_k$ is obtained from a dataset and is known for the entire prediction period.

At the first time step, $k = 0$, the input vector is initialized with values for $u$ and $y$ from a dataset. This is possible because the process is assumed to be in steady state before collecting data, as described in section 3.5.1. Accordingly the input vector for the first time step is

$$\mathbf{x}_0 = \begin{bmatrix} u_0 & \cdots & u_0 & y_0 & \cdots & y_0 \end{bmatrix} \tag{3.37}$$

### MC dropout-network

This section is rewritten version of section *MC dropout-network* in Eilertsen 2020. The MC dropout-network is an extension of the NARX network. This network implements the method for estimating epistemic uncertainty presented in section 2.6. A dropout layer, described in section 2.1.3, is added before every hidden layer of the NARX network. The same dropout rate is used for every dropout layer.

The network's prediction for one input is the average of $T$ forward passes of that input with different dropout configurations, as described in section 2.6. This prediction is denoted by $\hat{y}_{k+1}$. Hence, the input vector becomes

$$\mathbf{x}_k = (u_{k+1}, ..., u_{k-m_u+1}, \hat{y}_{k-m_y+1}, ..., \hat{y}_{k+1}) \tag{3.38}$$

The variance associated with each prediction is calculated using Equation (2.27), given in section 2.5.

### Full ADF network

The full ADF network is another extension of the NARX network. This network implements the method for estimating aleatoric uncertainty described in section 3.1. In this network, the linear layers which propagates intermediate values are replaced by linear layers, which propagate intermediate means and covariances. The same is done for the ReLU activation functions. The output mean and covariance of the linear layers are given by Equations (3.2a) and (3.2b). Moreover, the implementation of the calculation of the output mean and output covariance of the ReLU activation functions are given in section 3.2.2.

Since the network propagates means and covariances, the network's input vector can be described as

$$\mathbf{x}_k = \begin{bmatrix} \boldsymbol{\mu}_{x_k} & \boldsymbol{\Sigma}_{x_k} \end{bmatrix} \tag{3.39}$$

where $\boldsymbol{\mu}_{x_k}$ is a vector containing the mean values of the network's input at time step $k$. This vector can be expressed as

$$\boldsymbol{\mu}_{x_k} = \begin{bmatrix} u_k & \cdots u_{k-m_y} & \mu_{\hat{y}_{k-m_y}} & \cdots & \mu_{\hat{y}_k} \end{bmatrix} \tag{3.40}$$

where $u_k \cdots u_{k-m_y}$ are the process inputs at time $k \cdots k - m_u$ which are equal to process mean inputs at time $k \cdots k - m_u$ since the process inputs are assumed deterministic. $\mu_{\hat{y}_{k-m_y}} \cdots \mu_{\hat{y}_k}$ are the network's mean predictions at time steps $k \cdots k - m_y$. $\boldsymbol{\Sigma}_{x_k}$ is the inputs covariance matrix. This is a diagonal matrix due to the facts that the process inputs are assumed deterministic and that it is assumed that the network's predictions are independent. $\boldsymbol{\Sigma}_{x_k}$ can be expressed as

$$\boldsymbol{\Sigma}_{x_k} = \text{diag}\left(0, \cdots, 0, \sigma^2_{\hat{y}_k}, \cdots, \sigma^2_{\hat{y}_{k-m_y}}\right) \tag{3.41}$$

where $\sigma^2_{\hat{y}_k}, \cdots, \sigma^2_{\hat{y}_{k-m_y}}$ are the network's estimated aleatoric variances at time steps $k \cdots k - m_y$.

The output vector of the full ADF network is

$$\hat{\mathbf{y}}_k = \begin{bmatrix} \mu_{\hat{y}_k} & \sigma^2_{\hat{y}_k} \end{bmatrix} \tag{3.42}$$

where $\mu_{\hat{y}_k}$ and $\sigma^2_{\hat{y}_k}$ are the network's mean prediction and aleatoric variance estimate respectively.

The mean input vector is initialized at the first time step to the following

$$\boldsymbol{\mu}_{x_0} = \begin{bmatrix} u_0 & \cdots & u_0 & \cdots & \mu_{y_0} & \cdots & \mu_{y_0} \end{bmatrix} \tag{3.43}$$

where both $u_0$ and $\mu_{y_0}$ are assumed to be known.The network's initial input covariance matrix is

$$\boldsymbol{\Sigma}_{x_0} = \text{diag}\left(0, \cdots, 0, \sigma^2_{y_0}, \cdots, \sigma^2_{y_0}\right) \tag{3.44}$$

where $\sigma^2_{y_0}$ is the variance of the measurement of the process output.

### 3.5.3 Training

In this section, the training process of the MC-dropout NARX neural network and the full ADF NARX neural network is described. First the general training for both networks is described, then the specific for each network. The objective of the training is not to find the best networks, but to find networks which the uncertainty estimation methods can be evaluated on.

Both networks are trained as the following: a feedforward network has been trained to find the network parameters. Thereafter the validation loss was computed on a NARX neural network to evaluate the network's performance on data not seen during training. The MSE described by Equation (2.9) in section 2.1.3 has been used as the cost function during training and to find the validation performance. The network was trained with mini-batch gradient decent and back-propagation to calculate the gradients, both presented in section 2.1.3. Further, the ADAM-optimizer with learning rate $1 \times 10^{-3}$ was used to update the network parameters (Kingma and Ba 2017).

#### MC-dropout

The MC-dropout network has been trained on the first training set, which contains parts of the state space described in section 3.5.1. The hyperparameter tuning framework Ray Tune was used during training (Liaw et al. 2018). The framework trains several different versions of the network, with different hyperparameters concurrently. Where the hyperparameters are the mini-batch size, number of layers, number of units in each layer, dropout rate, number of previous process inputs, $m_u$, and number of previous process outputs, $m_y$. This framework was used to speed up the training process. The

final MC-dropout NARX neural network has the following architecture $m_u = 3$, $m_y = 3$, 3 hidden layers with 9 units each. The dropout rate used to train the network was 0.01.

#### Full ADF

Two different full ADF NARX networks have been trained, both networks have the same architecture, the same number of hidden layers and number of units in each layer. The networks have been trained on the second training set presented in 3.5.1. L2-regularization, described in section 2.1.3, was used to prevent overfitting. Moreover, several different combinations of number of previous process inputs and outputs where tried. The combination which provided the best results, in terms of lowest validation loss, were one previous process input and three previous process outputs, $m_u = 1$ and $m_y = 3$.

The final networks which have been used to test the uncertainty estimation methods have the following network architecture; two hidden layers with 3 units each. Both the networks have approximately the same validation loss, in the range of $5 \times 10^{-5}$ to $6 \times 10^{-5}$. Moving forward the networks will be denoted by network 1 and network 2.

### 3.5.4 Evaluation

The predictive performance of the networks will be evaluated with the MSE given in section 2.1.3 by Equation (2.9). The MSE will be calculated against the mean output of the process. The estimated variance of the full ADF networks will be compared between the two networks. The epistemic uncertainty estimated by MC-dropout will be compared between test cases.

# 4 Experiments and Results

In this chapter, the results from several experiments are presented. First the estimation of aleatoric uncertainty in an ARX network, presented in section 3.3 is tested together with the mechanism for resetting the estimated variance presented in the same section. Then, the method for estimating aleatoric variance in a neural network presented in section 3.4. Further, aleatoric uncertainty estimation in a NARX neural network, presented in 3.5.2 is tested. And finally estimation of epistemic uncertainty in a NARX neural network, presented in section 3.5.2 is tested.

## 4.1 Experiment 1: Estimating aleatoric uncertainty of a linear system

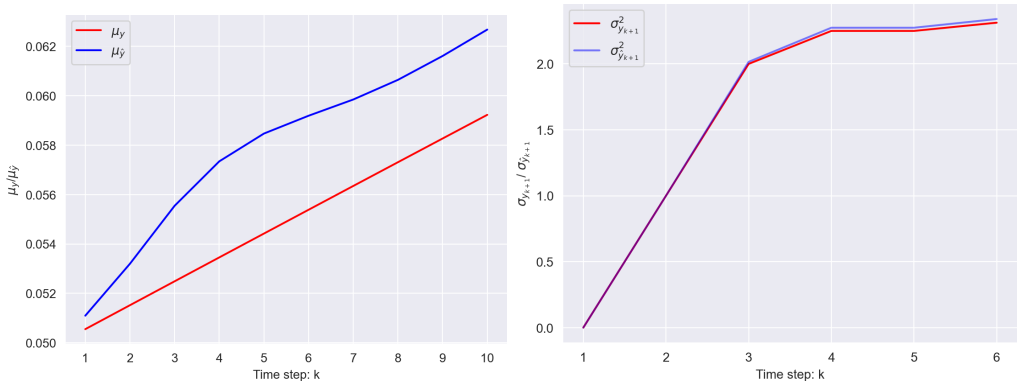
This experiment tests the estimation of aleatoric uncertainty in an ARX network presented in 3.3. First the the result from the uncertainty estimation is presented, then the result from resetting the estimated variance with a Kalman filter is presented. A test set have been created with the ARX model, described by Equation (3.20), which both cases are tested on.

### 4.1.1 Aleatoric uncertainty estimation

Figure 4.1 shows a visualization of the network's mean predictions together with the ground-truth mean, given by Equation (3.23). Further, the figure shows the estimated variance as well as the ground-truth variance. The ground-truth variance is obtained from Equation (3.24). Note that the mean predictions is shown for 10 time steps and that the variance estimation is shown for 6 time steps. This is done to capture the most essential parts of the mean predictions and variance estimation. The mean predictions starts by deviating from the ground-truth mean for time steps 1 to 5, where the deviation increases for each time step. Then at time step 6 the deviation decreases and for the rest of the predictions there is a constant error between the ground-truth mean and the predicted mean. The ground-truth mean is lower than the predicted mean for all time steps. The estimated variance for time steps 1 to 3 tracks the ground-truth variance perfectly. Then for prediction 4 and from there on, there is a constant error between the ground-truth and estimated variance, where the estimated variance is greater than the ground-truth. Table 4.1 summarizes the network's mean predictive and variance estimation performance. The network's mean prediction performance is better than the

network's estimated variance performance, in terms of the MSE against the ground-truths.

The deviations seen in both the mean predictions and variance estimate is due to the training set the ARX network is trained on. Since a disturbance is affecting every example in the training set, the network does not learn the underlying ARX model, which is not affected by a disturbance. The network instead learns to model the best fit of the training data. The reason for the mean predictions having a smaller deviations from the ground-truth than the variance estimate is most likely to due to the fact that the variance has a greater magnitude than the mean.




(a) Network's mean prediction, $\mu_{\hat{y}}$ and the ground truth mean $\mu_y$, given by Equation (3.23).

(b) Network's estimated variance, $\sigma^2_{\hat{y}}$ and the process ground truth variance $\sigma^2_y$, given by Equation (3.24).

Figure 4.1: Estimated variance of $\hat{y}$. The red plot is the ground-truth variance, $\sigma^2_y$, and the blue is the network's prediction, $\sigma^2_{\hat{y}}$. The ground-truth variance is obtained from equation (3.24). Note that the mean predictions are shown for 10 time steps and that the variance estimation is shown for 6 time steps. The mean predictions starts by deviating from the ground-truth mean for time steps 1 to 5, where the deviation increases for each time step. Then at time step 6 the deviation decreases and for the rest of the predictions there is a constant error between the ground-truth mean and the predicted mean. The ground-truth mean is lower than the predicted mean for all time steps. The estimated variance for time steps 1 to 3 tracks the ground-truth variance perfectly. Then for prediction 4 and from there on there is a constant error between the ground-truth and estimated variance, where the estimated variance is greater than the ground-truth.

### 4.1.2 Resetting the estimated variance with a Kalman filter

In this experiment the resetting of the full ADF ARX network estimated variance with a Kalman filer, described in section 3.3.5 is tested. A measurement of the ARX model is

| $\text{MSE}(\mu_{\hat{y}}, \mu_y)$ | MSE $(\sigma^2_{\hat{y}}, \sigma^2_y)$ |
|---|---|
| $9.71 \times 10^{-5}$ | $3.65 \times 10^{-4}$ |

Table 4.1: Results experiment 1: where $\text{MSE}(\mu_{\hat{y}}, \mu_y)$ is the mean prediction error and MSE $(\sigma^2_{\hat{y}}, \sigma^2_y)$ is the estimated variance error.

available every fourth time step. This section will only focus on the estimated variance, because the resetting of the variance with the Kalman filter has no effect on the mean predictions, since the mean predictions and estimated variance are independent in an full ADF ARX network. Thus, the network's mean predictions are the same as in the previous experiment. However, a visualization of the mean predictions is shown for completeness.

Figure 4.2 shows a visualization of the network's mean predictions in the same plot as the ground-truth mean, given by Equation (3.23). Moreover, the figure shows the network's estimated variance together with the ground-truth variance, given by Equation (3.24). One can see that every time a measurement from the ARX model is available that the estimated variance decreases. This is expected since new information about the system is available. The estimated variance is lower after the first measurement of the ARX model is available than when the rest of the measurements are available. This is due to the fact that estimated variance associated with previous prediction is lower when the first measurement is available then for the rest of the predictions. Moreover, the estimated variance is lower than the ground-truth variance for all predictions. This is because the measurements are available before the network reaches maximum estimated variance.

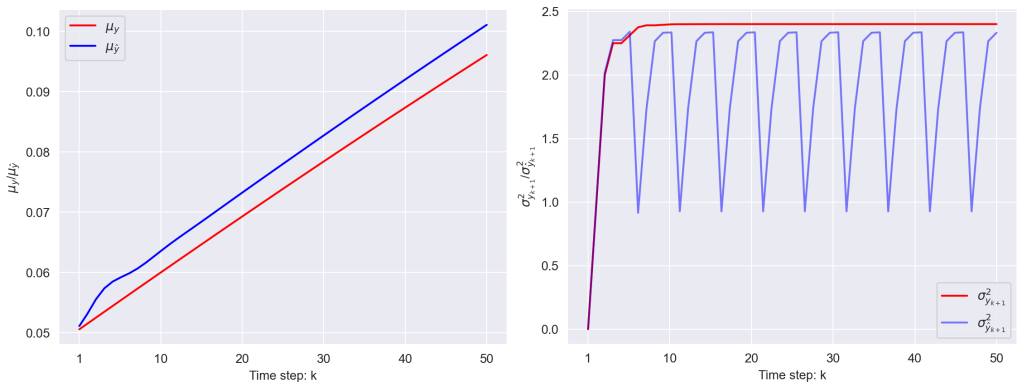
## 4.2 Experiment 2: Estimating aleatoric uncertainty in a nonlinear system

In this section, the two full ADF networks presented in section 3.4 are tested on the simple nonlinear system presented in the same section. The difference between the two networks are that on is trained on normalized data and the other is trained on data which is not normalized. Two test cases are used to evaluate the networks. The test cases have the same input means, but different input variances. The input mean is on the interval $[5, 10]$, and the inputs variances are as follows, 1 and 0.25.

### 4.2.1 Unnormalized network

In this section, the results from the network trained without normalized training data is presented.

Figures 4.3 and 4.4 shows visualization of the network's predictive performance and vari-




(a) Network's mean prediction, $\mu_{\hat{y}}$ and the ground truth mean $\mu_y$, given by Equation (3.23).

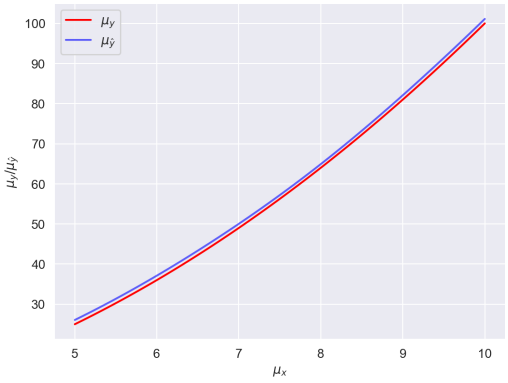
(b) Network's estimated variance, $\sigma^2_{\hat{y}}$ and the process ground truth variance $\sigma^2_y$, given by Equation (3.24). When the estimated variance is reset with a Kalman filter every fourth time step.

Figure 4.2: Result experiment 1: The estimated variance decreases every time a measurement from the ARX model is available.. Furthermore, the estimated variance is lower after the first measurement of the ARX model is available than when the rest of the measurements are available. Moreover, the estimated variance is lower than the ground-truth mean for all predictions.

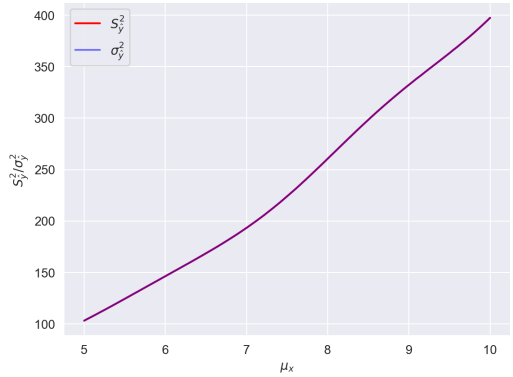
ance estimate when the input variance, $\sigma^2_x$, is equal to 1 and 0.25 respectively. Table 4.2 summarize the network's predictive and variance estimation performance with different input variances. Figures 4.3a and 4.4a shows the full ADF network's predicted mean, $\mu_{\hat{y}}$ in blue, together with the ground truth mean, $\mu_y$, in red, when the network has different input variances. The predictions are better, closer to the ground truth mean, in the test cases where the input variance is lower. In the test case where the input variance is 1 the network's mean prediction is greater than the ground truth mean for all predictions. These results show that the input variance has an affect on the network's predicted mean, which is expected since the output mean of the ReLU activation function is dependent on its input variance, which again is dependent on the network's input variance.

Furthermore, figures 4.3b and 4.4b shows the network's estimated output variance, $\sigma^2_{\hat{y}}$, in blue together with the network's output sample variance, $S^2_{\hat{y}}$, in red, for each input variance. One can see that the full ADF network's variance estimation very closely tracks the output sample variance for the tested input variances. Thus, the full ADF network consisting of linear layers and ReLU activation functions estimates the output variance of the network to be very similar to the true output variance of the network. Since the reference output variance of the network is found by sampling, which is not equal, but very similar, to the true output variance of the network, it is hard to quantify how similar the estimated variance is to the true variance.

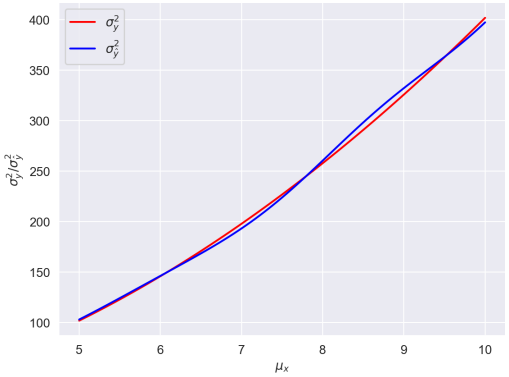
The system's output variance, $\sigma^2_y$, in red is shown together with network's estimated




(a) Network's mean prediction, $\mu_{\hat{y}}$ and the ground truth mean $\mu_y$, given by Equation (3.30).

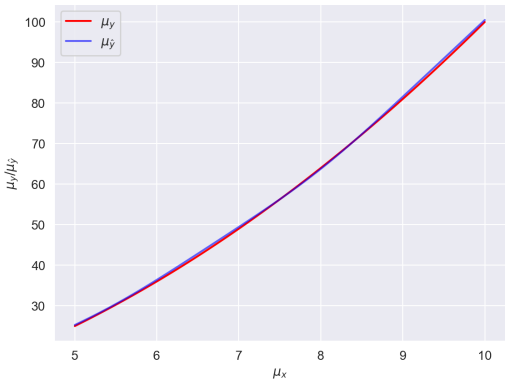



(b) Network's estimated variance, $\sigma^2_{\hat{y}}$ and the network's sample output variance $S^2_{\hat{y}_k}$.

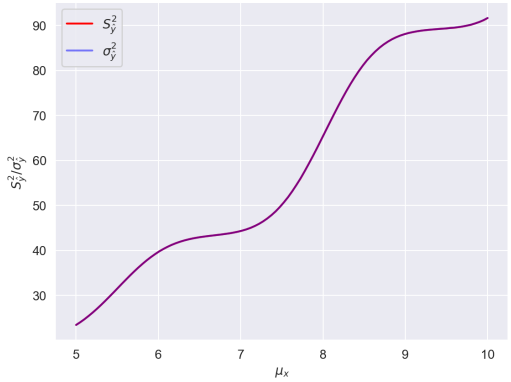



(c) Network's estimated variance, $\sigma^2_{\hat{y}}$ and the system's ground truth variance $\sigma^2_y$, given by Equation (3.31).

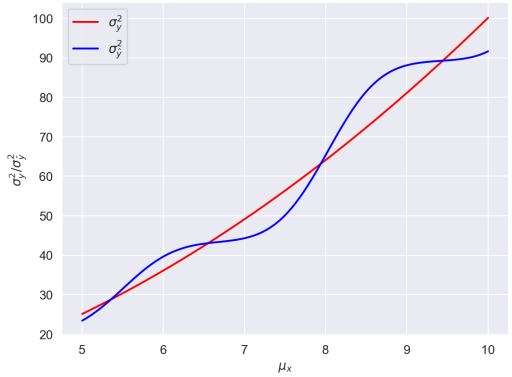
Figure 4.3: The network's predictive performance and variance estimate when the input variance, $\sigma^2_x$, is equal to 1 when the network is trained on data without normalization. The networks predictive mean is slightly greater than the ground-truth mean for all predictions. The estimated output variance is nearly equal to the output sample variance for all predictions. Moreover, the estimated output variance oscillates around the ground-truth output variance. Where the deviations from the ground-truth increase when the input mean increase.




(a) Network's mean prediction, $\mu_{\hat{y}}$ and the ground truth mean $\mu_y$, given by Equation (3.30).




(b) Network's estimated variance, $\sigma^2_{\hat{y}}$ and the network's sample output variance $S^2_{\hat{y}_k}$.




(c) Network's estimated variance, $\sigma^2_{\hat{y}}$ and the system's ground truth variance $\sigma^2_y$, given by Equation (3.31).

Figure 4.4: The network's predictive performance and variance estimate when the input variance, $\sigma^2_x$, is equal to 0.25 when the network is trained on data without normalization. The networks predictive mean is slightly greater than the ground-truth mean for all predictions. The estimated output variance is nearly equal to the output sample variance for all predictions. Moreover, the estimated output variance oscillates around the ground-truth output variance. Where the deviations from the ground-truth increase when the input mean increase.

output variance, $\sigma^2_{\hat{y}}$, in blue for each input variance in figures 4.3c and 4.4c. It is clear that the output variance of the network is not equal to the output variance of the system that it is approximated when the input variance is 0.25. The estimated variance oscillates around the system's output variance, increasingly deviating from the system's output variance as the mean input increases. The same can be seen when the output variance is 1, but to a lesser extent.

Full ADF does estimate the output variance in a neural network. However, this output variance is not the same as the output variance of the system that the network is approximating. Moreover, the estimated variance is not closer to the system's output variance if the network makes mean predictions closer to the ground truth mean, than if the predictions are poorer.

| $\sigma_x^2$ | MSE($\mu_{\hat{y}}, \mu_y$) | MSE ($\sigma^2_{\hat{y}}, \sigma^2_y$) | MSE ($\sigma^2_{\hat{y}}, S^2_{\hat{y}}$) |
|---|---|---|---|
| 1 | 1.110 | 14.96 | $1.40 \times 10^{-3}$ |
| 0.25 | 0.192 | 24.14 | $8.15 \times 10^{-5}$ |

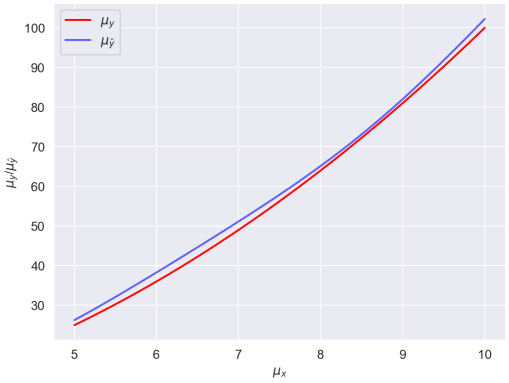
Table 4.2: Predictive and variance estimation performance on the network trained on training data without normalization with different input variances.

### 4.2.2 Normalized network

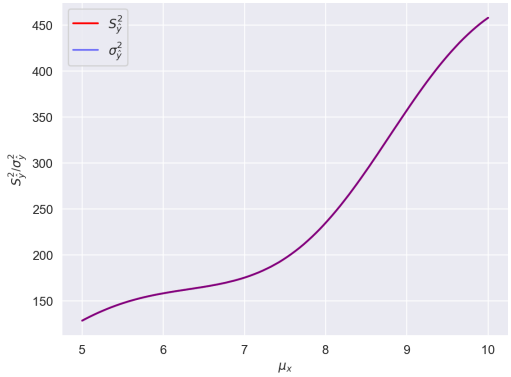
In this section, the results from the network trained with normalized training data is presented.

Figures 4.5 and 4.6 shows visualization of the network's predictive performance and variance estimate when the input variance, $\sigma_x^2$, is equal to 1 and 0.25 respectively. Table 4.3 summarize the network's predictive and variance estimation performance with different input variances. Figures 4.5a and 4.6a shows the full ADF network's predicted mean, $\mu_{\hat{y}}$ in blue, together with the ground truth mean, $\mu_y$, in red, when the network has different input variances. The network's mean predictive performance is slightly better when the input variance is 1 than when it is 0.25. The mean prediction for each input variance has the same shape, it over predicts the output mean from input mean equal to 5 to approximately 8. Then from input mean 8 to 10 the network still over predict the mean when the input variance is equal to 1, but to a lesser extent. When the input variance is 0.25 the network slightly under predicts the output mean on the same interval.

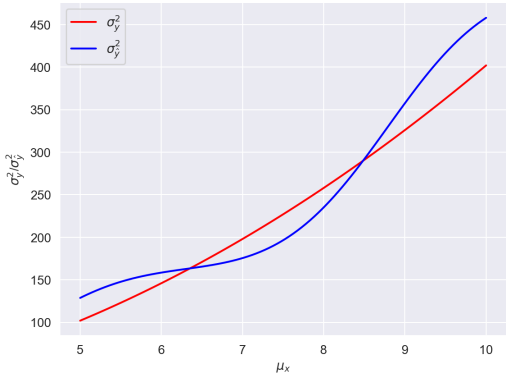
Moreover, figures 4.5b and 4.6b shows the network's estimated output variance, $\sigma^2_{\hat{y}}$, in blue together with the network's output sample variance, $S^2_{\hat{y}}$, in red, for each input variance. One can see that the full ADF network estimates the output variance to be very similar to the output sample variance for the tested input variances. Hence, the full ADF network consisting of linear layers and ReLU activation functions estimates the output variance of the network to be very similar to the true output variance of the network. Since the reference output variance of the network is found by sampling, which




(a) Network's mean prediction, $\mu_{\hat{y}}$ and the ground truth mean $\mu_y$, given by Equation (3.30).



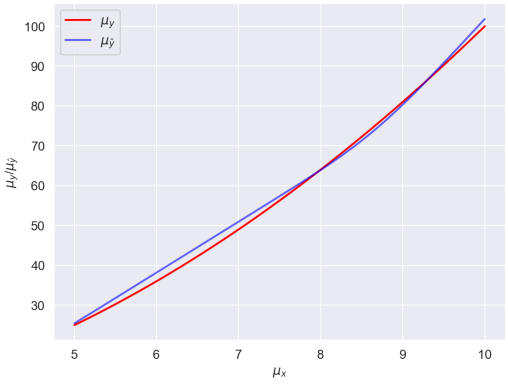

(b) Network's estimated variance, $\sigma^2_{\hat{y}}$ and the network's sample output variance $S^2_{\hat{y}_k}$.



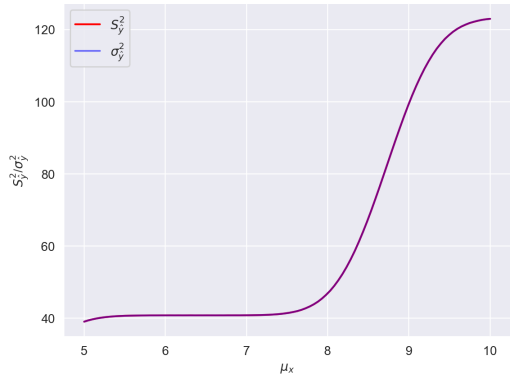

(c) Network's estimated variance, $\sigma^2_{\hat{y}}$ and the system's ground truth variance $\sigma^2_y$, given by Equation (3.31)

Figure 4.5: The network's predictive performance and variance estimate when the input variance, $\sigma^2_x$, is equal to 1 when the network is trained on normalized data. The estimated output variance is nearly equal to the output sample variance for all predictions. Moreover, the estimated output variance oscillates around the ground-truth output variance. Where the deviations from the ground-truth increase when the input mean increase.

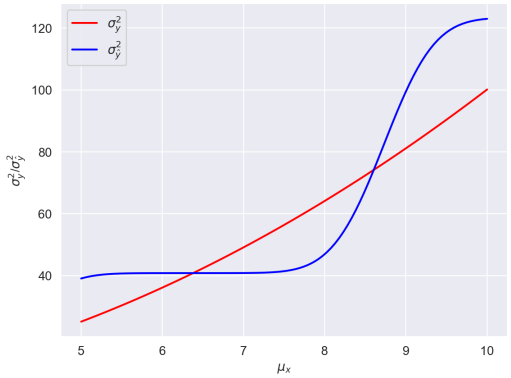
is not equal, but very similar to the true output variance of the network, it is hard to quantify how similar the estimated variance is to the true variance.




(a) Network's mean prediction, $\mu_{\hat{y}}$ and the ground truth mean $\mu_y$, given by Equation (3.30).




(b) Network's estimated variance, $\sigma_{\hat{y}}^2$ and the network's sample output variance $S^2$.




(c) Network's estimated variance, $\sigma_{\hat{y}}^2$ and the network's sample output variance $S_{\hat{y}_k}^2$. $\sigma_y^2$, given by Equation (3.31).

Figure 4.6: The network's predictive performance and variance estimate when the input variance, $\sigma_x^2$, is equal to 0.25 when the network is trained on normalized data. The estimated output variance is nearly equal to the output sample variance for all predictions. Moreover, the estimated output variance oscillates around the ground-truth output variance. Where the deviations from the ground-truth increase when the input mean increase

The system's output variance, $\sigma_y^2$, in red is shown together with network's estimated output variance, $\sigma_{\hat{y}}^2$, in blue for each input variance in figures 4.5c and 4.6c. When the input variance is 0.25 the full ADF network estimates an almost constant variance from input mean equal to 5 to approximately 8. Then the estimated variance rapidly increases. In both the test cases the estimated variance oscillates around the true output variance of the system.

| $\sigma_x^2$ | MSE($\mu_{\hat{y}}, \mu_y$) | MSE ($\sigma_{\hat{y}}^2, \sigma_y^2$) | MSE ($\sigma_{\hat{y}}^2, S_{\hat{y}}^2$) |
|---|---|---|---|
| 1 | 2.061 | 227 | $1.63 \times 10^{-3}$ |
| 0.25 | 3.023 | 874 | $1.02 \times 10^{-4}$ |

Table 4.3: Predictive and variance estimation performance on the network trained on normalized data different input variances.

Full ADF does estimate the output variance in a neural network which is trained on normalized data. However, this output variance is not the same as the output variance of the system that the network is approximating.

## 4.3 Experiment 3: Estimating aleatoric uncertainty in a NARX neural network

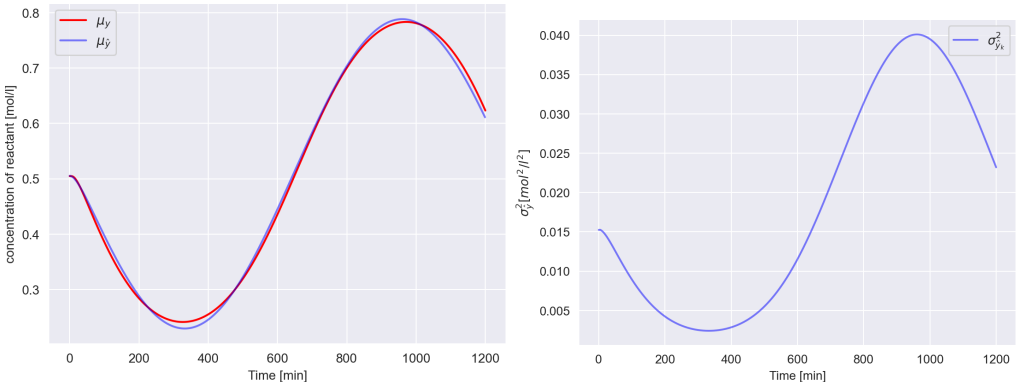
In this experiment, the full ADF NARX neural network presented in section 3.5.2 is tested. The two full ADF NARX networks, network 1 and network 2 introduced in section 3.5.3 are tested on the same test set. The test set is created by applying a time varying input signal to the CSTR-process, described in section 3.5.1, which covers the whole state space.

The networks initial variance associated with the process outputs used as network inputs at time step zero is equal to the noise added to process outputs in the training set, which is $\sigma_{y_0}^2 = 0.01525^2\,\text{mol}^2/\text{l}^2$. The process input is not affected by noise. The full ADF networks initial input covariance matrix is therefore given by

$$\mathbf{\Sigma}_{x_0} = \text{diag}\left(0, \cdots, 0, \sigma_{y_0}^2, \cdots, \sigma_{y_0}^2\right) \tag{4.1}$$

Figure 4.7 shows a visualization of the mean predictions of network 1 together with the ground-truth mean and the network's estimated output variance for each prediction. One can see that the mean predictions follows the ground-truth mean closely for nearly all predictions. Table 4.4 summarizes the network's predictive performance. The network's estimated variance approximately follows the same shape as the mean predictions, in the sense that estimated variance is lower when the mean prediction is lower and the estimated variance is greater when the mean predictions are greater. One can see that the estimated variance increase and decrease more rapidly when ground-truth mean and predictive mean is higher, than is does when the ground-truth mean and predictive mean is lower. This can be seen in the intervals between approximately 800 min to 1175 min and approximately 200 min to approximately 425 min respectively. Furthermore, the estimated variance is greater than the initial input variance for all predictions.

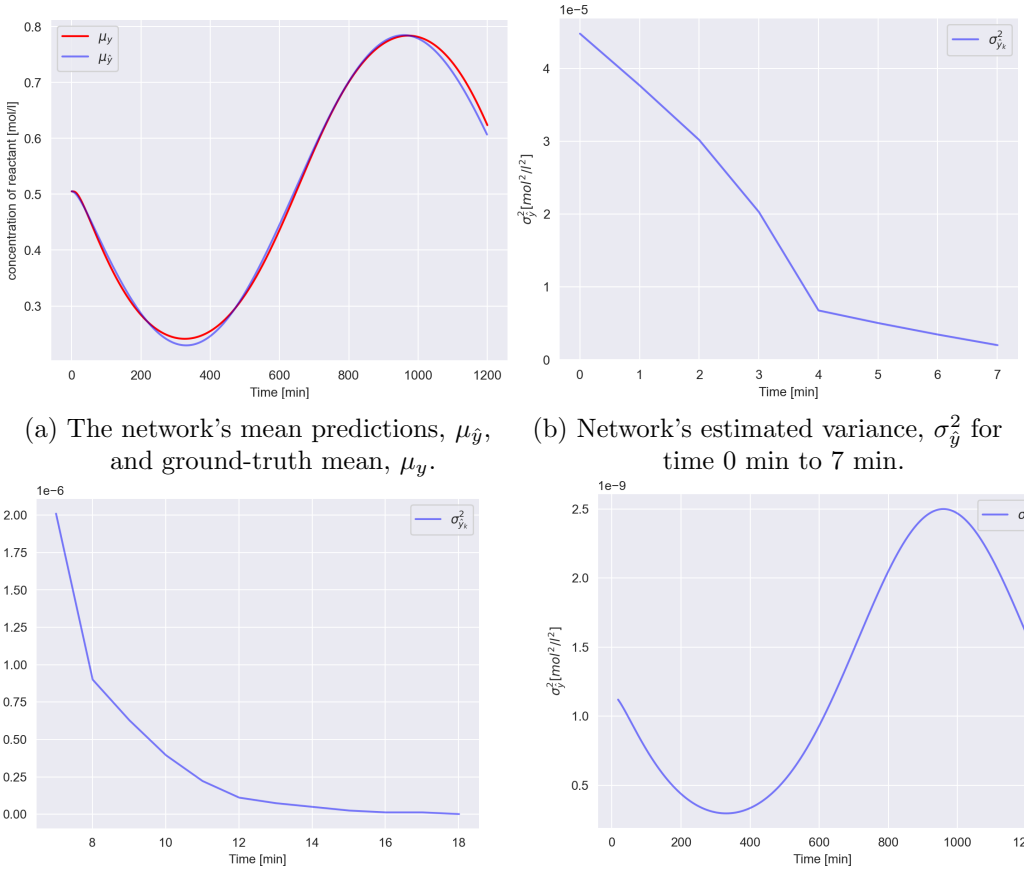
Figure 4.8 shows a visualization of network 2's mean predictions together with the ground-truth mean, and the network's estimated output variance for each prediction. Note that the variance estimate is shown in three separate figures for better visualization.




(a) The network's mean predictions, $\mu_{\hat{y}}$, and ground-truth mean, $\mu_y$.

(b) The estimated variance associated with each prediction, $\sigma^2_{\hat{y}}$.

Figure 4.7: Results experiment 3 network 1: The mean predictions follows the ground-truth mean closely for nearly all predictions. The network's estimated variance approximately follows the same shape as the mean predictions, in the sense that estimated variance is lower when the mean prediction is lower and the estimated variance is greater when the mean predictions are greater. Furthermore, the estimated variance increase and decrease more rapidly when ground-truth mean and predictive mean is higher, than is does when the ground-truth mean and predictive mean is lower. This can be seen in the intervals between approximately 800 min to 1175 min and approximately 200 min to approximately 425 min respectively.




(a) The network's mean predictions, $\mu_{\hat{y}}$, and ground-truth mean, $\mu_y$.

(b) Network's estimated variance, $\sigma^2_{\hat{y}}$ for time 0 min to 7 min.

(c) Network's estimated variance, $\sigma^2_{\hat{y}}$ for time 7 min to 18 min.

(d) Network's estimated variance, $\sigma^2_{\hat{y}}$ for time 18 min to 1200 min.

Figure 4.8: Results experiment 3 network 2: The mean predictions tracks the ground-truth mean closely for nearly all predictions. The estimated variance begins by rapidly decreasing for each prediction before settling at time 18 min. From that point forward the network's estimated variance approximately follows the same shape as the mean predictions, in the sense that estimated variance is lower when the mean prediction is lower and the estimated variance is greater when the mean predictions are greater. Furthermore, the estimated variance increase and decrease more rapidly when ground-truth mean and predictive mean is higher, than is does when the ground-truth mean and predictive mean is lower. This can be seen in the intervals between approximately 800 min to 1175 min and approximately 200 min to approximately 425 min respectively.

The mean predictions tracks the ground-truth mean closely for nearly all predictions. Table 4.4 summarizes the network's predictive performance. The estimated variance begins by rapidly decreasing for each prediction before settling at time 18 min. And from that point forward the estimated variance has the same shape as the variance estimated in network 1, but with a lower magnitude. In this network the estimated variance is lower than the input variance for all predictions.

From the results from network 1 and network 2 it is clear that two networks with the same architecture, same number of layers with the same number of units in each layer which is approximating the same system with approximately equal mean predictive performance, estimates the variance to be completely different in order of magnitude. However, the form of the estimated variance is similar in the two networks, in the sense that both networks estimates lower variance for the same inputs. Moreover, the network which has the greatest performance in terms of the MSE has a higher estimated variance than the network with poorer performance.

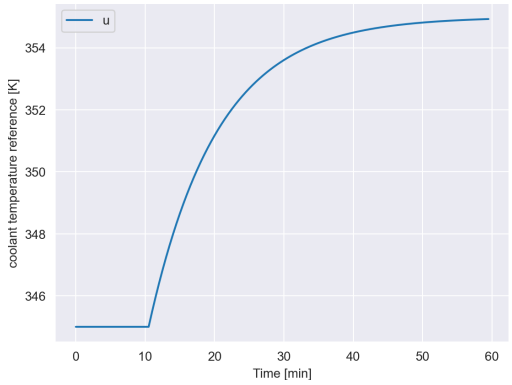
| Network | MSE |
|---|---|
| 1 | $5.96 \times 10^{-5}$ |
| 2 | $6.88 \times 10^{-5}$ |

Table 4.4: Results experiment 3: both networks mean predictive performance.

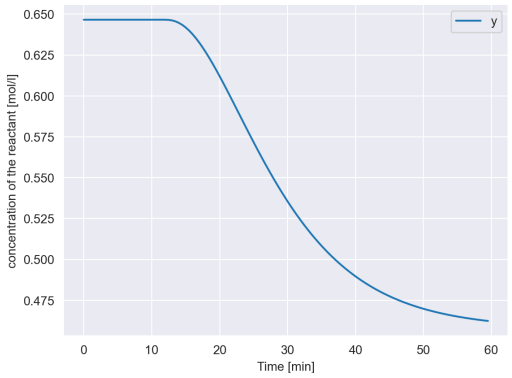
## 4.4 Experiment 4: Estimating epistemic uncertainty in a NARX neural network

This experiment is the same as section *4.2 Experiment 2: model uncertainty* in Eilertsen 2020. In this experiment epistemic uncertainty estimation with MC-dropout, described in section 2.6, is tested. The experiment is performed on the MC-dropout network, presented in section 3.5.2. To evaluate the network's model epistemic capabilities, two test sets are used, one which is generated from a CSTR-process (presented in section 3.5.1) with an input signal in the range of $345\,\mathrm{K} < u < 355\,\mathrm{K}$, which is inside the training space, introduced in section 3.5.1. This test set will be denoted by inside the training space. The second test set is generated from a CSTR-process input signal in the range $335\,\mathrm{K} < u < 345\,\mathrm{K}$, which is outside the training space, and accordingly, this test set will be denoted by outside the training space. Both input signals have the same form and cover equal parts of the input space, but because the CSTR-process is nonlinear, the output signals have the same form, but do not cover equal parts of the output space. The input and output signals for both test sets are visualized in Figure 4.9. One can see that the output from the CSTR-process in the test set inside the training space, covers a larger part of the output space than the output signal in the test set outside the training space.

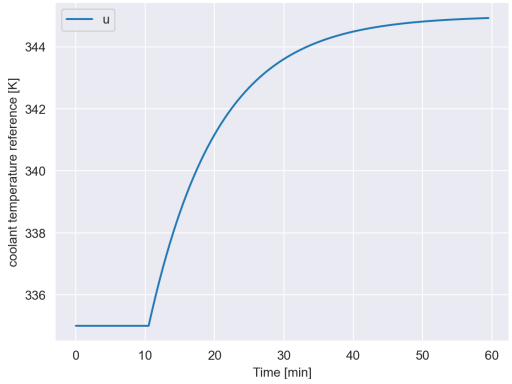
For every input, 200 samples are used to obtain the network's prediction and associated



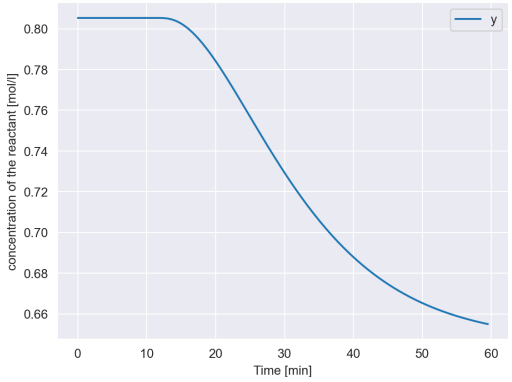

(a) CSTR-process input for test set inside training space




(b) CSTR-process output for test set inside training space

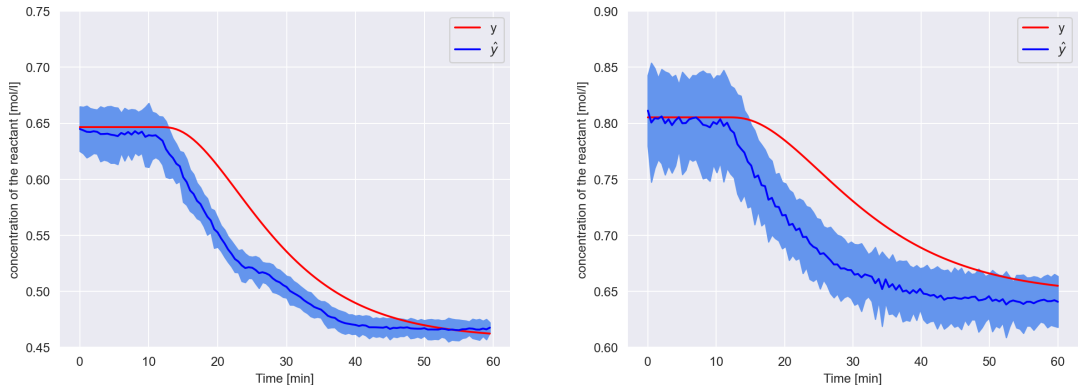



(c) CSTR-process input for test set outside training space




(d) CSTR-process output for test set outside training space

Figure 4.9: Input and outputs from the CSTR-process of both test sets used in experiment 1. The input signals for both tests cover an equal part of the input space, while the process output signals do not cover equal parts of the output space. The output signal in the test set inside the training space covers a larger part of the output space than the output signal in the test set outside the training space.




(a) inside training space

(b) outside training space

Figure 4.10: Results experiment 4. The red plot is the ground-truth signal, $y$, and the blue is the network's prediction, $\hat{y}$, The shaded blue area is the predictions $\pm$ one standard deviation. The estimated one standard deviation is higher in the test set outside the training space, and the predictions are closer to the ground-truth in the test set inside the training space. Furthermore, the estimated one standard deviation is higher when the process is at steady state than for the rest of the predictions for both test sets. The predictions are closer to the ground-truth in the test set outside the training space when the process is at steady state.

epistemic uncertainty on both test sets. Figure 4.10 shows a visualization of the predictions and the ground-truth signal together with the predictions estimated one standard deviation on both test sets. From this figure, it is clear that the estimated one standard deviation is higher on the test set outside the training space than on the test set inside the training space. Furthermore, the predictions on the test set inside the training space are closer to the ground-truth than the predictions on the test set outside the training space. Moreover, the estimated uncertainty, when the process is in a steady state is higher than when it is not, for both test sets. The performance on the test set outside the training space is better than on test set inside the training space, when the process is at steady state.

The higher degree of uncertainty associated with the predictions when the process is at steady state can be explained by the training set, which is presented in section 3.5.1. All the examples in the training set are time varying process input signals. Therefore, the process at steady state is not seen during training, and the predictions made when the process is at steady state will have a higher degree of uncertainty associated with them. Moreover, the test set outside the training space has a higher estimated variance for predictions when the process is at steady than the test set inside the training space. So the uncertainty estimated is affected by at the minimum two factors in the training set. The first one is that if the predictions are made on data in the same range as the data in the training set, and the second if the process input signal frequency is represented

in the training set.

On the other hand, the reason why the predictive performance on the test set outside the training space is better than the performance on the test set inside the training space, when the process is at steady state, cannot be explained by the training set. One would expect that the predictive performance on the test set inside the training space would be better because the predictions are made on data in the range seen by the network during training, but that is not the case. Since there are no constant process input signals in the training set, the network is not trained to preform predictions on such signals. So it can be a coincidence that the performance on the test set outside the training set is better. If another constant input signal had been used in one or both test sets, the outcome might have been different. The performance on the test set inside the training space could have been better than the performance on the test set outside of the training space.

Table 4.5 summaries the network's performance and corresponding uncertainty estimates on both test sets. From these results, it is clear that the estimated uncertainty is higher on the test set outside the training space, which is in line with the theory presented in section 2.6.

| - | MSE | $v_{min}$ | $v_{max}$ | $\bar{v}$ |
|---|---|---|---|---|
| Inside training space | $9.00 \times 10^{-4}$ | $2.64 \times 10^{-5}$ | $8.00 \times 10^{-4}$ | $2.00 \times 10^{-4}$ |
| Outside training space | $1.80 \times 10^{-3}$ | $3.00 \times 10^{-4}$ | $2.80 \times 10^{-3}$ | $1.00 \times 10^{-3}$ |

Table 4.5: Results experiment 4: $v_{min}$ is the lowest estimated variance, similarly $v_{max}$ is the highest estimated variance, and $\bar{v}$ is the average estimated variance, all in $mol^2/l^2$.

## 4.5 Experiment 2: Estimating epistemic uncertainty in a NARX neural network

This experiment is the same as section *4.2 Experiment 2: model uncertainty* in Eilertsen 2020. In this experiment epistemic uncertainty estimation with MC-dropout, described in section 2.6, is tested. The experiment is performed on the MC-dropout network, presented in section 3.5.2.

A test set is created by applying an input signal to the CSTR-process, described by equation (3.32), which covers the whole state space. For every input, 200 samples are used to obtain the network's prediction and associated model uncertainty. Figure 4.11 shows a visualization of the predictions and the associated estimated one standard deviation together with the ground-truth signal. The green shaded area in the figure shows the part of the output space, which was not included in the training space. The figure shows that the estimated variance is higher in the region not included in the

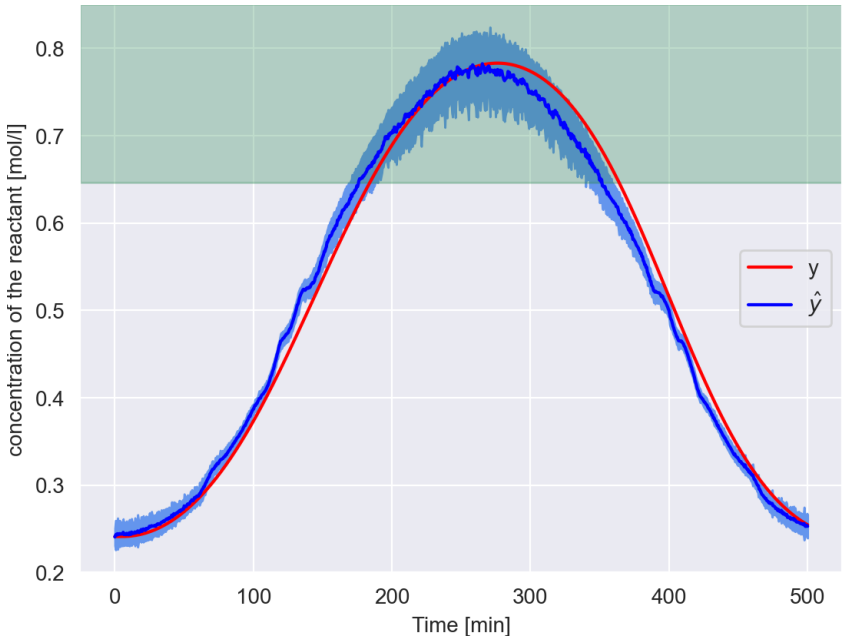



Figure 4.11: Results experiment 5. The green shaded area is not included in the training space. The red plot is the ground-truth signal, $y$, and the blue is the network's prediction, $\hat{y}$, and the blue shaded area is the predictions $\pm$ one standard deviation. The estimated one standard deviation is higher in the area not included in the training space. Furthermore, the prediction made in the middle of the training space has a lower degree of uncertainty associated with them than other predictions within the bounds of the training space. The network's performance is generally better inside the training space, but the network achieves its best performance in the first half of the predictions outside of the training space. This performance is not sustained for the second half of the prediction outside of the training space.

training space. Furthermore, the estimated one standard deviation is lower when the predictions are made in the middle of the training space, specifically between 0.55 mol/l and 0.3 mol/l, than when it is not and still within the bounds of the training space. The deviations between the predictions and the ground-truth are generally smaller in the training space. However, the first half of the prediction outside of the training space has a very low deviation from the ground-truth. But this deteriorates in the second half of the prediction that is outside the training space. And these predictions deviate the most from the ground-truth.

The higher degree of uncertainty associated with certain predictions inside the training space can be explained by the fact that part of the training space is underrepresented in the training set. The training set is presented in section 3.5.1. So, in addition to estimating a higher degree of uncertainty associated with predictions outside the training space, predictions made on data that is less represented in the training set have a higher degree of uncertainty associated with them. This is in line with the theory presented in section 2.6. The uncertainty estimate is higher for prediction made on data which is underrepresented or not represented in the training set. Moreover, the estimated uncertainty scales with how represented it is in the training set. The uncertainty associated with the predictions made outside the training space is higher than the uncertainty associated with the predictions made on the data which is present in the training set, which is underrepresented. Table 4.6 summaries the network's performance and corresponding uncertainty estimate.

| MSE | $v_{min}$ | $v_{max}$ | $\bar{v}$ |
|---|---|---|---|
| $3.83 \times 10^{-4}$ | $1.72 \times 10^{-5}$ | $2.60 \times 10^{-3}$ | $5.00 \times 10^{-4}$ |

Table 4.6: Results experiment 5: $v_{min}$ is the lowest estimated variance, similarly $v_{max}$ is the highest estimated variance, and $\bar{v}$ is the average estimated variance, all in $mol^2/l^2$.

# 5 Discussion and Further Work

In this section the results from the experiments will be discussed and further work will be suggested. The discussion regarding experiment 4 and 5 are rewritten versions of the discussion of the results given in Eilertsen 2020.

## 5.1 Discussion

Experiment 1 shows that full ADF estimates the aleatoric uncertainty in an ARX network which approximates an ARX model. There are some discrepancies between the ground-truth mean and the estimated mean, and the ground-truth variance and the estimated variance. However, this is expected because the network is trained on data which is affected by a disturbance. If however the ARX network where to approximate a linear dynamical system instead of an ARX model the outcome might be different. When a dynamical system is approximated there are more parameters which effect the network's model parameters, which affect the estimation of aleatoric uncertainty. Most importantly are the number of previous system inputs, $m_u$ and network outputs, $m_y$, which are used to obtain the next prediction. For any given linear dynamical system a number of combinations of $m_u$ and $m_y$ will yield a network with good predictive performance. Yet it is not certain that any of these network's captures the uncertainty aspects of the dynamical system. Furthermore, considering that the dynamical system will be assumed unknown to the network, there exists no method to verify that the network's estimated variance is the same as the variance of the dynamical system.

Furthermore, experiment 1 shows that a Kalman filter, where the state-transition model and control input model is given by the model parameters in the ARX network, can be used to reset the estimated variance, without any knowledge about the underlying system that the ARX network is approximating. As with the ARX network, for this to work one must find the model parameters which captures the uncertainty aspect of the underlying system.

Experiment 2 demonstrates that full ADF estimates the output variance of a neural network to be very similar to the sample variance of the network, regardless of whether the network has been trained with normalized data or not. Considering that the true output variance of the system is intractable, it is difficult to determinate how close to the true output variance the full ADF variance estimation is, although, the sample output variance provides a good indication. Moreover, it is not assumed that full ADF can exactly estimate the output variance in a network consisting of linear layers and

ReLU activation functions, because the calculation of the covariance output of a ReLU activation function contains approximations.

Moreover, experiment 2 shows that a neural network's estimated variance is not the same as the output variance of the system the neural network is approximating. A neural network is not an exact replica of the system it is approximating. It is a combination of linear transformations and nonlinear activation functions. Furthermore, the network is only an approximation on the interval contained in the network's training data. Thus, the network does not approximate the underlying system well or at all outside of this interval. Therefore, the network only learns the system on the interval it is trained on and not the system as a whole.

Furthermore, experiment 2 shows that the estimated output variance for both networks oscillates around the ground-truth output variance of the system the networks are approximating. So there is at least some connection between the variance of the networks and the system. There probably exists some realizations of the networks which have the same output variance as the system, but there are as of now no way of finding them. However, in general the output variance of a neural network is not equal to the output variance of the system which the network is approximating.

The variance estimates made by the network trained without normalized data is better than the estimates made by the other network, when compared to the output variance of the system that is approximated by the networks. Furthermore, the predictive performance show a similar result, the predictive performance is poorer on the network trained with normalized data. There is no way of telling if this is due to the normalization or because of chance. In general when a network is trained with normalized data, the predictive performance is better, the network is more stable and it is easier to train because most training procedures are tailored to normalized data. Normalization especially make the training process easier when the network has inputs which have different orders of magnitude.

Experiment 3 demonstrates that full ADF can be used to estimate the output variance in a NARX neural network when it is assumed that subsequent predictions are independent. Further, the experiment shows that two networks with the same network architecture which have approximately the same mean predictive performance, yields vastly different output variance estimates in terms of order of magnitude. However, the form of the estimates are the same, in the sense that both networks estimates higher and lower variance for the same predictions.

Clearly the estimated output variance is not the true output variance of the NARX neural network since subsequent predictions are not independent. It is however the best estimation which is currently possible. Moreover, the true output variance of the network is intractable, so it is hard to evaluate how good the estimates are and how much the cross correlations between subsequent predictions affect the true output variance of a NARX neural network. But based on the results from experiment 2 and the results from the different networks in experiment 3, which respectively indicate that the estimated

output variance and the sample variance in a neural network oscillates around the true output variance of the system being approximated, and that the estimated variance in the two NARX neural networks have significantly differences in order of magnitude. It is probable that the cross correlations between the consecutive predictions have a major impact on the true variance of the network, because if the estimated variance in a NARX neural network was to oscillate around the true variance such a great discrepancy between the estimated variance in the two networks would not have been observed.

The estimation of the output covariance of the ReLU activation function is computationally expensive, since the bivariate probability has to be calculated between all the elements in the functions input vector. In the implementation used in this thesis these are calculated one at a time. This renders full ADF unsuitable for large networks where the time aspect is important.

The results from experiments 4 and 5 show that MC-dropout can be used to estimate epistemic uncertainty in NARX neural networks. The uncertainty estimates are higher for predictions which are either underrepresented or not represented in the training set. Furthermore, the uncertainty estimates scale with how represented the data on which the prediction is made is in the training set. The estimated uncertainty associated with situations which are not represented in the training set is higher than the uncertainty associated with situation which there are few examples of in the training set.

It is difficult to quantify the uncertainty estimates since there exists no ground-truth for epistemic uncertainty. The exact values which are estimated might not be so important, it is the estimated uncertainty for one prediction compared to the estimated uncertainty for other predictions that is important. If the MC-dropout method were to be used in practice a baseline uncertainty estimate which is associated with predictions made on situations that are well represented in the training set can be found. Then the estimated uncertainties can be compared to the baseline to decide the level of uncertainty.

Experiment 4 shows that the network's performance is better on data seen by the network during training. One the other hand, experiment 5 does not show the same, the network's best performance is on data not seen during training, but the network's worst performance is also on data not seen during training. As the network used to obtain the results is not trained for optimal performance these results might suggest that a higher degree of uncertainty does not indicate that the network's prediction is poorer than when the uncertainty estimate is lower. So, the uncertainty estimates only indicated when the network is preforming predictions on situations it has not seen during training or have seen few examples of during training. Then to utilize the uncertainty estimate the network must be trained for optimal performance, which will most likely provide better performance in situations where the estimated uncertainty is higher and vice versa. The uncertainty estimate cannot be used as a replacement for training the network optimally.

The MC-dropout network that have been used to obtain the results in experiment 4 and 5 has been trained with a small dropout rate equal to 0.01. Moreover, there is no

noise in the training set. Since dropout is a regularization method used to prevent the network from modeling the noise in the training set, a small dropout rate can be used, and dropout is only used to estimate the uncertainty, and not for regularization. In other applications where the training set contains noise a higher dropout rate must be used for regularization. According to the theory presented in section 2.6 the best dropout rate for estimating the uncertainty is the same as the dropout rate the network is trained with. This suggest that the uncertainty estimation is not sensitive to the dropout rate.

## 5.2 Further Work

As the estimation of aleatoric uncertainty in an ARX network with a Kalman filter to reset the estimated uncertainty is only tested on an ARX model, it would be interesting to see if the method proposed could be used to estimate the aleatoric uncertainty in a linear dynamical system, estimated with an ARX model.

Moreover, a method for estimating the cross-correlation between subsequent predictions in a NARX neural network is needed to estimate the true output variance of a NARX neural network.

# 6 Conclusion

This work attempts to estimate the uncertainty in ARX networks and NARX neural networks. The aleatoric uncertainty estimating method ADF is extended to the case where it is assumed that both the network inputs and the outputs from each network layer are multivariate Gaussian distributed, this method is denoted by full ADF. Full ADF has been tested on an ARX network, a neural network and a NARX neural network. In, addition has MC-dropout to estimate epistemic uncertainty been tested on a NARX neural network.

Full ADF estimates the aleatoric uncertainty in an ARX network when the ARX network is approximating an ARX model. Furthermore, can the model parameters of the ARX network be used as the system equation in a Kalman filter to reset the network's estimated variance when measurements from the ARX model is available.

In a neural network estimates full ADF the output variance of the network to be very similar to the output sample variance of the network, which is the best estimate of the true output variance that is available. Hence, the estimated variance obtained with full ADF in a neural network is a good estimate of the true output variance of the network. Moreover, the estimated variance is not the same as the output variance of the system which is approximated by the neural network. However, the estimated variance oscillates around the output variance of the system, thus there exists a relationship between the estimated variance and the true variance of the system that is approximated by the neural network.

Full ADF can also be used to obtain an estimate of the output variance in a NARX neural network. However, this estimate is not close to the true output variance of the NARX neural network, because there exists not method for estimating the cross correlations between subsequent predictions made by the network which are used as inputs in the next predictions.

MC-dropout can be used to epistemic model uncertainty in NARX neural networks. MC-dropout estimates a higher degree of uncertainty associated with predictions made on situations which are underrepresented in the training set. Furthermore, the estimated uncertainty scales with how represented the situation is in the training set. It can be difficult to quantify the estimated uncertainty.

# Bibliography


Alanazi, Mohana, Mohsen Mahoor, and Amin Khodaei (2017). *Day-Ahead Solar Forecasting Based on Multi-level Solar Measurements*. arXiv: 1710.03803 [cs.CE].

Boyen, Xavier and Daphne Koller (1998). "Tractable Inference for Complex Stochastic Processes". In: *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence—UAI 1998*. Available at http://www.cs.stanford.edu/~xb/uai98/. San Francisco: Morgan Kaufmann, pp. 33–42.

Damianou, Andreas and Neil Lawrence (29 Apr–01 May 2013). "Deep Gaussian Processes". In: *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Carlos M. Carvalho and Pradeep Ravikumar. Vol. 31. Proceedings of Machine Learning Research. Scottsdale, Arizona, USA: PMLR, pp. 207–215. URL: http://proceedings.mlr.press/v31/damianou13a.html.

Eilertsen, I. (2020). *Uncertainty estimation in nonlinear autoregressive exogenous neural networks*.

Evans, M.J. and J.S.Rosenthal (2004). *Probability & Statistics - The Science of Uncertainty*. New York: W.H.Freeman.

Gal, Yarin (2016). "Uncertainty in Deep Learning". PhD thesis. University of Cambridge.

Gal, Yarin and Zoubin Ghahramani (2015). *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. arXiv: 1506.02142 [stat.ML].

Gast, Jochen and Stefan Roth (2018). *Lightweight Probabilistic Deep Networks*. arXiv: 1805.11327 [cs.CV].

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Harville, David A. (1997). "The Moore-Penrose Inverse". In: *Matrix Algebra From a Statistician's Perspective*. New York, NY: Springer New York, pp. 497–519. ISBN: 978-0-387-22677-4. DOI: 10.1007/0-387-22677-X_20. URL: https://doi.org/10.1007/0-387-22677-X_20.

Hinton, Geoffrey E. and R. Neal (1995). *Bayesian learning for neural networks*.

Ioffe, Sergey and Christian Szegedy (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv: 1502.03167 [cs.LG].

Kalman, Rudolph Emil (1960). "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* 82.1, p. 35. DOI: 10.1115/1.3662552. URL: http://dx.doi.org/10.1115/1.3662552.

Kendall, Alex and Yarin Gal (2017). *What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?* arXiv: 1703.04977 [cs.CV].

Kingma, Diederik P. and Jimmy Ba (2017). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG].

Kullback, S. and R. A. Leibler (Mar. 1951). "On Information and Sufficiency". In: *Ann. Math. Statist.* 22.1, pp. 79–86. DOI: 10.1214/aoms/1177729694. URL: https://doi.org/10.1214/aoms/1177729694.

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell (2017). *Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles*. arXiv: 1612.01474 [stat.ML].

Lauritzen, Steffen Lilholt (1992). "Propagation of probabilities, means and variances in mixed graphical association models". English. In: *Journal of the American Statistical Association* 87, pp. 1098–1108. ISSN: 0162-1459.

Liaw, Richard et al. (2018). "Tune: A Research Platform for Distributed Model Selection and Training". In: *arXiv preprint arXiv:1807.05118*.

Loquercio, Antonio, Mattia Segu, and Davide Scaramuzza (Apr. 2020). "A General Framework for Uncertainty Estimation in Deep Learning". In: *IEEE Robotics and Automation Letters* 5.2, pp. 3153–3160. ISSN: 2377-3774. DOI: 10.1109/lra.2020.2974682. URL: http://dx.doi.org/10.1109/LRA.2020.2974682.

MacKay, David J. C. (1992). "A Practical Bayesian Framework for Backpropagation Networks". In: *Neural Computation* 4.3, pp. 448–472. DOI: 10.1162/neco.1992.4.3.448. eprint: https://doi.org/10.1162/neco.1992.4.3.448. URL: https://doi.org/10.1162/neco.1992.4.3.448.

Manzano, J. M. et al. (2019). "Output feedback MPC based on smoothed projected kinky inference". In: *IET Control Theory Applications* 13.6, pp. 795–805. DOI: 10.1049/iet-cta.2018.5522.

Maybeck, Peter S. (1979). *Stochastic models, estimation and control / Peter S. Maybeck*. English. Academic Press New York. Chap. 12.7, 3 v. : ISBN: 0124807011 012480702 0124807038.

Minka, Thomas P. and Rosalind Picard (2001). "A Family of Algorithms for Approximate Bayesian Inference". AAI0803033. PhD thesis. USA.

Mitchell, Tom M. (1997). *Machine Learning*. New York: McGraw-Hill. ISBN: 978-0-07-042807-2.

Nair, Vinod and Geoffrey Hinton (June 2010). "Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair". In: vol. 27, pp. 807–814.

Narendra, K. S. and K. Parthasarathy (1990). "Identification and control of dynamical systems using neural networks". In: *IEEE Transactions on Neural Networks* 1.1, pp. 4–27. DOI: 10.1109/72.80202.

Opper, Manfred and Ole Winther (Jan. 1999). "A Bayesian approach to on-line learning". English. In: *On-line learning in neural networks*. Ed. by David Saad. Publications of the Newton Institute. Copyright of Cambridge University Press Available on Google Books. United Kingdom: Cambridge University Press, pp. 363–378. ISBN: 0262194163. DOI: 10.2277/0521652634.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-

pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Postels, Janis et al. (2019). *Sampling-free Epistemic Uncertainty Estimation Using Approximated Variance Propagation*. arXiv: 1908.00598 [cs.LG].

Rahimi, Zhoobin, Helmi Zulhaidi Mohd Shafri, and Masayu Norman (2018). "A GNSS-based weather forecasting approach using Nonlinear Auto Regressive Approach with Exogenous Input (NARX)". In: *Journal of Atmospheric and Solar-Terrestrial Physics* 178, pp. 74–84. ISSN: 1364-6826. DOI: https://doi.org/10.1016/j.jastp.2018.06.011. URL: http://www.sciencedirect.com/science/article/pii/S1364682618302220.

Rosenbaum, S. (1961). "Moments of a Truncated Bivariate Normal Distribution". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 23.2, pp. 405–408. ISSN: 00359246. URL: http://www.jstor.org/stable/2984029.

Ruder, Sebastian (2017). *An overview of gradient descent optimization algorithms*. arXiv: 1609.04747 [cs.LG].

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, pp. 318–362. ISBN: 026268053X.

Seborg, Dale E., Thomas F. Edgar, and Duncan A. Mellichamp (Aug. 1989). *Process Dynamics and Control*. New York: Wiley.

Socci, Nicholas, Daniel Lee, and H. Sebastian Seung (1998). "The Rectified Gaussian Distribution". In: *Advances in Neural Information Processing Systems*. Ed. by M. Jordan, M. Kearns, and S. Solla. Vol. 10. MIT Press. URL: https://proceedings.neurips.cc/paper/1997/file/28fc2782ea7ef51c1104ccf7b9bea13d-Paper.pdf.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

Tang, Liyang (2020). *Application of Nonlinear Autoregressive with Exogenous Input (NARX) neural network in macroeconomic forecasting, national goal setting and global competitiveness assessment*. arXiv: 2005.08735 [physics.soc-ph].

Tavares, Lucas A., Petrus E. O. G. B. Abreu, and Luis A. Aguirre (2020). *Nonlinearity Compensation Based on Identified NARX Polynomials Models*. arXiv: 2011.12246 [eess.SY].

Taylor, John R. (1996). *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements*. 2 Sub. University Science Books. ISBN: 093570275X. URL: http://www.amazon.com/Introduction-Error-Analysis-Uncertainties-Measurements/dp/093570275X%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D093570275X.

Van Rossum, Guido and Fred L. Drake (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace. ISBN: 1441412697.

Virtanen, Pauli et al. (2020). "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.