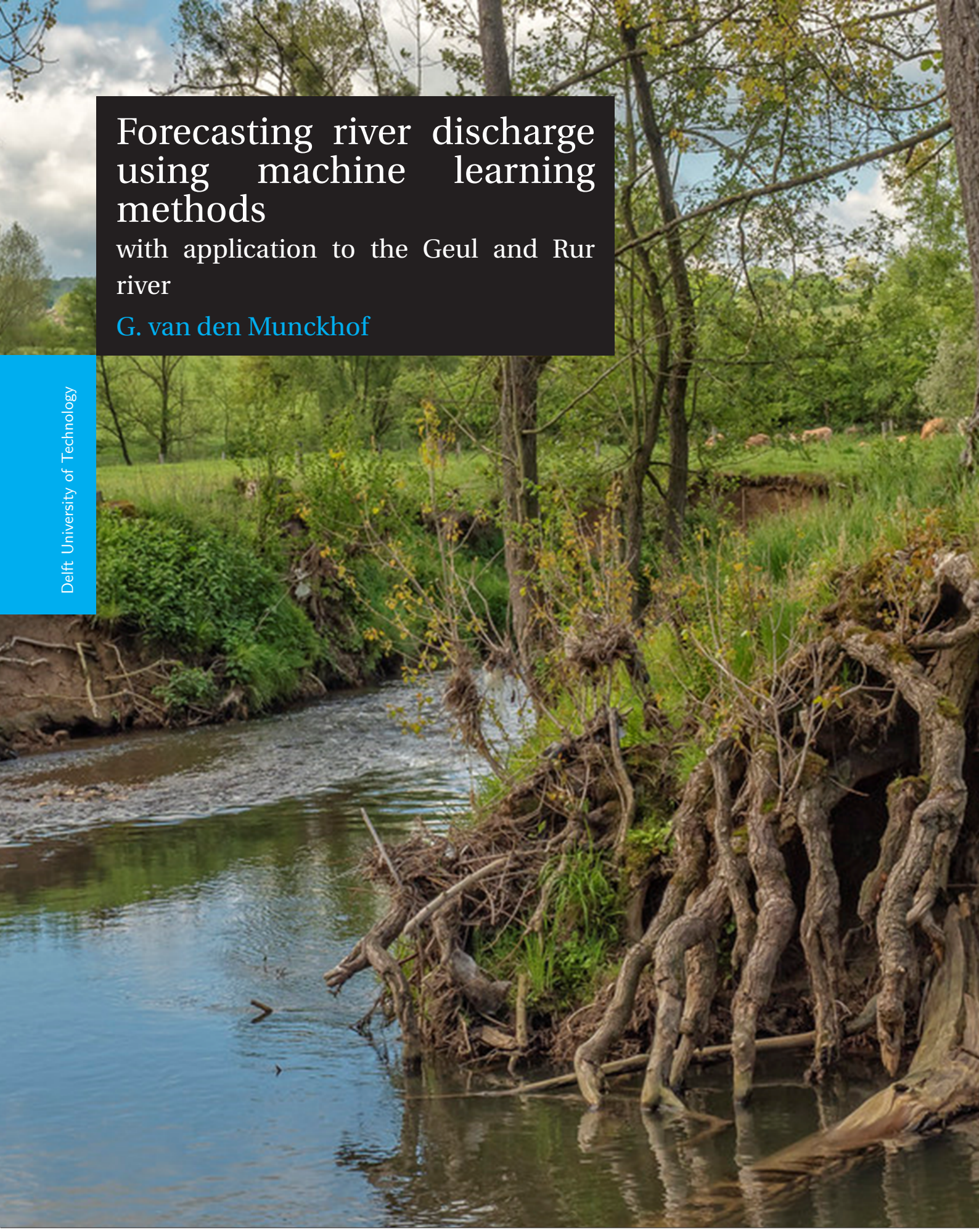

Forecasting river discharge using machine learning methods

with application to the Geul and Rur river

G. van den Munckhof

Delft University of Technology

TU Delft Delft University of Technology

Challenge the future

FORECASTING RIVER DISCHARGE USING MACHINE LEARNING METHODS

WITH APPLICATION TO THE GEUL AND RUR RIVER

by

G. van den Munckhof

in partial fulfillment of the requirements for the degree of

Master of Science
in Civil Engineering

at the Delft University of Technology,
to be defended publicly on Thursday March 12, 2020 at 2:30 PM.

Student number:	4694767	
Thesis committee:	Dr. M. Hrachowitz (chair)	Delft University of Technology
	Dr. E. Ragno	Delft University of Technology
	Dr. ir. O. Morales Napoles	Delft University of Technology

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft Delft University of Technology

PREFACE

This report is my final product for the master degree in Water Management at Delft University of Technology. The thesis concerns the use of machine learning to forecast discharge, applied to two catchments which are partly in my beloved province of Limburg, southern Netherlands. As I would like to apply science to an actual problem, I wrote my thesis at Waterschap Limburg, one of the many water boards in the Netherlands.

Firstly, I would like to thank my thesis committee for their advice, feedback and guidance. Especially my daily supervisor Elisa Ragno for being patient and taking the time to supervise me. Secondly, I would like to thank Sabine Bartusseck for guiding me at the water board. Your practical input made sure the science served a purpose. Also, thanks to the employees of the water board in general for their interest in this subject and all the information shared.

G. van den Munckhof
Delft, March 2020

ABSTRACT

The objective of this study is find out whether maximum daily discharge of the Geul and Rur catchments can be forecast using machine learning (ML) methods, and if so, to what extent. In addition, these ML models are compared to a conceptual model to see which performs better. A second objective is to test whether soil moisture content (SMC) and NDVI increase performance of the two ML models.

The Geul and Rur catchments are both partly situated in the administrative area of Waterschap Limburg, a water authority in the Netherlands. They use discharge forecasts in order to prepare flood defenses and to monitor high water levels more closely. Currently, discharge is forecast using the conceptual HBV model for the Geul. Forecasting is done based on experience for the Rur and only in case of high water levels.
Conceptual and physical models are based on physical laws, i.e. conservation of mass and energy. However, some relations are not yet fully understood, or are hard to translate to equations, and assumptions have to be made. This is why a data-driven approach is used, as no explicit relationship between variables has to be specified.

In this study a gradient boosted decision tree framework (XGBoost) and a recurrent deep learning model (LSTM) are used to map input to output. XGBoost is a relatively new framework that has shown promising results in other water resources related studies. Long Short-Term Memory is a type of recurrent neural network and chosen for its ability to handle long-term dependencies and for its ability to model non-linearities. In order to see whether the machine learning methods outperform conceptual models, they are compared to the GR4J model. GR4J is a simple yet effective soil moisture accounting model. Beside SMC and NDVI, meteorological variables are used as input.

Results show that the deep learning model performs best for simulating today's discharge and when forecasting up to three days ahead. The GBDT model has a slightly higher Nash-Sutcliffe Efficiency (NSE) for the daily simulation of the Geul, but also a higher mean absolute error (MAE) compared to the deep learning model. The same holds for the three-day-ahead forecast for the Geul and the Rur. Peak timing is accurate for most models but peak discharge is often underestimated.
When comparing the ML models to the conceptual model for the daily simulation, deep learning performs best in terms of MAE, but GBDT is better in terms of NSE. When looking at the one-day-ahead forecast, deep learning outperforms the GBDT and conceptual model in both NSE and MAE. In any case, when looking at the metric they outperform the conceptual model.
However, the conceptual model has only a couple of parameters to calibrate, is transparent and has only two input variables. The ML models, on the other hand, have my parameters to train, are difficult to physically interpret and have four to five input variables.

Besides comparing the two types of models, it is tested whether adding soil moisture content and NDVI as input improve performance of the machine learning models. The former undoubtedly improves performance, whereas NDVI at best improves performance as much as some other meteorological variables.

Overall, this study finds that a conceptual model still outperforms the two ML models from a holistic point of view. However, machine learning is not yet fully exploited in water resources management. It already gives promising results and is likely to perform even better in the future.

Keywords: data driven, machine learning, gradient boosting, neural networks, XGBoost, LSTM, discharge, forecasting

List of Figures

List of Tables

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
CPU	Central Processing Unit
ECMWF	European Centre for Medium-range Weather Forecasts
ESA-CCI	European Space Agency - Climate Change Initiative
GBDT	Gradient Boosted Decision Trees
GIS	Geographical Information Systems
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HBV	Hydrologiska Byrans Vattenbalansavdelning
KNMI	Koninklijke Nederlandse Meteorologisch Instituut
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
NDVI	Normalized Difference Vegetation Index
NIR	Near-Infrared Radiation
NOAA	National Oceanic and Atmospheric Administration
NSE	Nash-Sutcliffe Efficiency
ReLu	Rectified Linear unit
SMC	Soil Moisture Content
TPU	Tensor Processing Unit
VIS	Visible radiation
WL	Waterschap Limburg (Water board of Limburg)
XGBoost	eXtreme Gradient Boosting

Contents

1

INTRODUCTION

1.1. PROBLEM MOTIVATION & STATEMENT

According to the Royal Netherlands Meteorological Institute (KNMI), climate change will result in more extreme weather in the Netherlands [7]. Extreme precipitation events are more likely to occur and the amount of precipitation will increase. In turn, discharge peaks will be higher. On the other side of the spectrum, there will be less precipitation in the summer and droughts are more likely to happen [7]. Reliable discharge forecasts support many water resources applications such as environment protection, hydropower generation, irrigation planning, water supply and sustainable development of water resources [8].

Many rainfall-runoff models have been developed over the past decades to forecast discharge. Most of them can be distinguished by their model structure (empirical, conceptual or physical) and representation of spatial processes (lumped, semi-distributed or distributed) [9]. In recent years, however, data-driven models are becoming more popular. Besides new algorithms, main drivers behind data-driven models are data availability and computer processing power. Both have improved greatly in the past decades, explaining the increase in popularity of data-driven models. A tremendous amount of spatial data has become available through remote sensing techniques such as satellites and radars [10], and geographical information systems (GIS) have made it easy to process these huge quantities of raster and vector data [11]. Additionally, measurement instrumentation improved during the past decades. All kinds of hydrologic variables can be measured more accurately on-site, and data transmission can be done using telemetry [12]. To give an example, stream flow and groundwater level can be monitored continuously remotely. For all this data to be processed within a feasible amount of time, more computational power is required. The invention of the Tensor Processing Unit (TPU) by Google made it possible to process more data in less time. A TPU is 15 to 30 times faster than its contemporary CPU or GPU [13]. To give an impression, one TPU can process more than 100 million photos per day [14].

Conceptual and physical hydrologic models are based on physical laws, i.e. conservation of mass and energy. However, some relations are not yet fully understood or are hard to translate to equations. Assumptions have to made to fill gaps of knowledge, which is not ideal.
This is where the data-driven approach comes in. It is defined by the Oxford dictionary as follows: 'based on or decided by collecting and analysing data' [15]. Basically, it is a more objective approach to modelling. It can be argued that the data-driven modeller still has to specify a structure, but this structure is already one step more abstract from the actual field of application.

One data-driven approach is the use of machine learning (ML). Machine learning is a subfield of artificial intelligence concerned with automating systems, or algorithms, that learn from data, identify patterns and make predictions. Its origin goes back to the 1950s but is now more entangled in our lives than ever before. To give some examples, e-mail spam and malware filtering, online customer support, online fraud detection, product recommendations, virtual personal assistants and self-driving cars use ML [16].
There are many ML algorithms available nowadays for regression, such as support vector machines, random forests, artificial neural networks, gradient boosted decision trees (GBDT) and naive Bayes to name a few. In

this thesis a GBDT framework (XGBoost) and a recurrent neural network (LSTM) is implemented.
XGBoost is relatively new as the paper was published in 2016 [3]. However, it has already found some application in the field of water management. It has been used for estimating reference evaporation [17, 18], forecasting daily discharge [19], urban pipe-break prediction [20], water stress [21], water quality monitoring [22], identification of urban drinking water supply patterns [23], retrieval of total precipitable water [24], and streamflow simulation [25]. XGBoost, and GBDT models in general, have many advantages; tree models are easily interpretable; are fast to train; can deal with continuous and categorical data; automatically deal with missing data; are quite robust to outliers; can capture non-linear relationships; do not require normalization or scaling; scale well to larger data sets and give the importance of each variable [26, 27]. It is for these reasons that this framework is chosen for this study. Of course, this type of model also has some limitations, tree models tend to select predictors with more distinct values; lack smoothness; have difficulty capturing additive structures and are prone to overfit if predictors have many categories [26, 28–30].

Contrary to XGBoost, LSTM models have been around for quite some years. In water management they have been used for predicting water quality [31–33], predicting discharge [34, 35], flood forecasting and simulations [36–38], daily reservoir inflow forecasting [39], real-time reservoir operation [40], downscaling rainfall [41], forecasting precipitation [35, 42, 43], predicting water table depth [44], lake level prediction [45] and sewer management [46, 47]. LSTM models, and artificial neural networks (ANNs) have many advantages, they: are robust to noisy data and measurement errors; adapt over time as the training data change; are excellent at modelling non-linear behaviour; can be trained on multiple processors to reduce computational time; and do not assume any distributions, eliminating errors in parameter estimation [48]. The main reason an LSTM model is implemented in this thesis is because it can model non-linear behaviour and because it can handle a time dependence. However, ANNs have their limitations too: trained parameters do not represent anything physical; they do not show the importance or significance of the input parameters automatically; they generally need lots of data to train on; they are prone to overfitting the data; and there is no standardized way of selecting network architecture. This is often done based on the user's experience and preference [48].

1.1.1. SOCIETAL RELEVANCE

The thesis is conducted in cooperation with Waterschap Limburg (WL), which is one of the 21 water authorities in the Netherlands. Its administrative area coincides with the province of Limburg (2209 km^2), covers 33 counties and has approximately 1.1 million inhabitants. The public authority's annual budget is 160 million Euro, most of which collected through taxes. The water authority employs about 450 full-time equivalents. Water authoritys are responsible for managing water barriers, waterways, water levels, water quality and sewage water treatment. WL maintains 2700 km of waterways and 180 km of primary flood defences [49].
The Meuse river flows from the south of the administrative area all the way to the north. Several rivers empty into the Meuse in this area, among which the Geul river and the Rur river, see Figure 1.1. The former being the western one and the latter the eastern one.
The societal relevance of this thesis is best expressed through the water authority. WL uses the discharge of the Geul river and Rur river as boundary conditions to calibrate their hydrologic models, and as warning for high water levels in these tributaries and rivers downstream. Although the water authority continuously measures stage of these tributaries, they do not forecast discharge for both rivers continuously. The discharge of the Geul is forecast daily using the Hydrologiska Byråns Vattenbalansavdelning (HBV) model, the discharge of the Rur is forecast only in case of high water level. If this is the case, forecasting is based on experience as there currently is no model available to accurately forecast discharge.
Improving the discharge forecasts for these rivers can result in better decision making, which serves the inhabitants of the water authority's district. Also, discharge forecasts can warn for expected high water levels. This warning gives the water authority more time to act on the situation or to monitor the situation more carefully. To give an example, WL has 203 coupures [49]. A coupure is a means of closing an opening in a (flood)wall or dike. It is a way of allowing traffic to pass a flood protection structure. In case of high water levels, these temporarily flood defences have to be built manually. As this requires time, building teams have to be notified in advance before the water arrives at the specific coupure.

1.1.2. SCIENTIFIC RELEVANCE

Many studies to forecast discharge using machine learning techniques have been conducted in the past [8, 50–52]. However, as far as the author is aware, discharge forecasting using machine learning with as input meteorological variables, normalized difference vegetation index (NDVI) and soil moisture content (SMC)

has not been conducted yet.
One study investigated the effects of NDVI as a biophysical input along with hydro-climatic inputs on modelling performance. The study found a significant effect of NDVI inputs for estimating monthly run-off using neural networks [53]. It has been shown that SMC is an important state variable in operational hydrology [54], and that it is the principal factor that controls whether incident rainfall becomes surface runoff or infiltrates into the soil [55].
While positive results have been achieved in the past for modelling rainfall-runoff using Long Short-Term Memory models (LSTM), more research is needed to verify these results [56]. Another study found that meteorological variables (daily precipitation, min/max temperature, solar radiation and vapour pressure) combined with past discharge could predict runoff with accuracies comparable to the SAC-SMA + Snow-17 Model [34]. Therefore, it might be worth trying to increase accuracy even more by adding SMC and NDVI.

1.2. SCOPE

The scope of this thesis is to develop a forecast model for the Geul and Rur rivers by applying two machine learning methods; a recurrent neural network and a gradient boosted decision tree. The catchment of the Geul river is modelled up to the gauge in Hommerich, and the catchment of the Rur up to Stah. Figure 1.1 shows the catchments and where they are located. The discharge will be modelled on a hourly and daily resolution for the Geul, and for the Rur on a daily resolution.

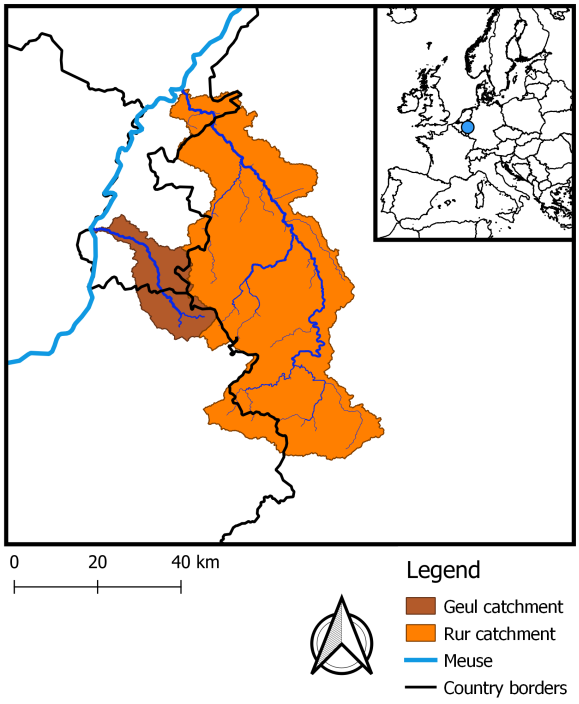

Figure 1.1: Location of the Geul and Rur catchments, both situated near the border of the Netherlands, Belgium and Germany.

1.3. RESEARCH QUESTION

The research question to be answered during this thesis is the following:

Is it possible to forecast discharge of the Rur and Geul river using a LSTM or GBDT model, and if so, to what extent?

In order to answer the research question, the following sub questions will be investigated:

- What is the performance for different time horizon forecasts?
- Which type of machine learning model is best suited for forecasting discharge?
- Do soil moisture content and NDVI improve model performance?
- Do the machine learning models outperform a conceptual model?

1.4. REPORT OUTLINE

This report consists of five chapters. The second chapter discusses the materials used. Information about the modelled catchments and variables used in the models can be found here. Chapter three treats the Gradient Boosted Decision Tree, Long Short-Term Memory model, and the GR4J conceptual model. It is explained how these models work, how data is preprocessed and how the models are trained. The results of both models are discussed and compared to the conceptual model in chapter four. The conclusion of this thesis, as well as directions for further research, can be found in chapter five.

There are two intermezzos in the third chapter. Here, concepts are explained more in depth. They are recognizable as the text is in between two horizontal lines.

2

MATERIALS

In this chapter materials used in this study are presented. Firstly, the two modelled catchments are described. Secondly, it is explained why certain variables are used and where the data comes from, including specifications about resolution. Lastly, as much of this thesis consisted of programming, used software is mentioned.

2.1. CATCHMENT DESCRIPTIONS

Two (sub)catchments are modelled: the Geul catchment and the Rur catchment. Both are situated in the Netherlands, Belgium and Germany, and can be found near the border triangle.

2.1.1. THE GEUL RIVER

The source of the Geul river is situated near the village of Lichtenbusch in the very east of Belgium. After 58 kilometres the Geul flows into the Meuse near the village of Meerssen. The drainage basin is 380 km^2, of which 52%, 42% and 6% is situated in the Netherlands, Belgium and Germany, respectively. It is one of few hill country rivers in the Netherlands with a relatively steep gradient as it covers 250 meters over 58 kilometres, see Figure 2.1 for an elevation map. The gradient varies from 0.02 m/m near the source to 0.0015 m/m at outflow. It is classified as a Cfb climate according to the Köppen climate classification system, meaning it is a temperate climate with a warm summer without dry season.
Average discharge of the Geul is 4 m^3/s but is highly variable between 0.8 m^3/s in the summer and 65 m^3/s during storms in the winter [57]. Annual average rainfall is 800 mm and distributed rather evenly throughout the year. It ranges from 45 mm per month in March to 75 mm per month in August. Rainfall intensities are generally low, but this is changing due to climate change. Rainfall in the summer is becoming more intense and concentrated [58]. Annual average evaporation is 500 mm resulting in a surplus of 300 mm a year, of which most is released during wintertime [57].
In the past the river had been straightened and riverbanks protected to prevent erosion. Beside reducing erosion, this also resulted in higher peak discharges. However, large floodings of the Meuse by the end of the twentieth century called for measures to reduce these peak discharges. At the beginning of this century, large stretches of the river have been given space to meander freely to a certain extent to combat this. In addition, water retention basins were constructed and areas were transformed into natures reserves to be inundated during high-water events. In 2004 the river was partly constricted and partly meandering. Currently, it is meandering within a certain meander zone, save some short stretches that flow through villages.
The Geul river is fed by surface water, shallow subsurface flow and groundwater flows. Rain recharges the groundwater (at 30-45 m below plateaus), flows horizontally along impenetrable clay layers and water springs originate where this flow meets the surface. Springs contribute significantly to base flow. Thus, the Geul is a rain fed river. As a consequence, discharge can change rapidly, e.g. during thunderstorms. [59]
The river's valley is incised in Mesozoic and Paleozoi shales, quartzites, limestones and sandstones. The main soil type is loess [60], and floodplains mainly consist of a two to three meter thick sequence of fine sand and loam [61]. The deposits are very cohesive resulting in steep banks.
Land cover wise, just over half of the catchment is used for agriculture and pastures. Forest and urban areas each take up 14%. The remainder is made up of industrial and commercial areas, mines, arable land and sport facilities. Figure 2.2 shows the land cover of both catchments, Table 2.1 shows the percentage of each

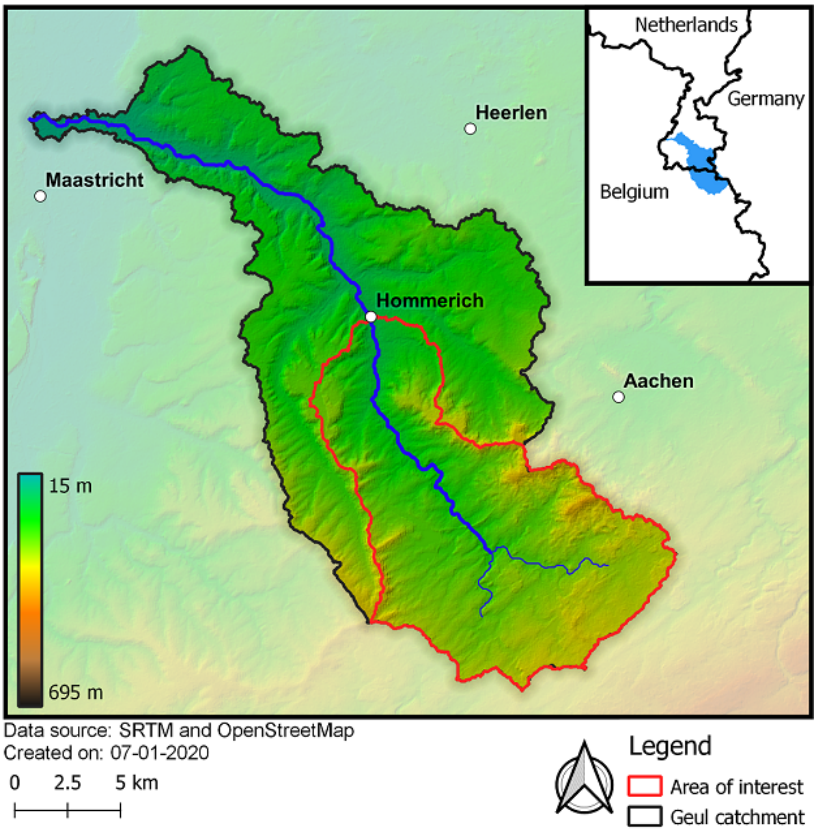

Figure 2.1: Elevation map of the Geul catchment. The gauge at Hommerich (centre of map) is modelled.

class. In the Dutch part of the catchment, heavy rain combined with bare agricultural fields and a relatively steep slope resulted in mud flows in 2018 [62].

As mentioned in the introduction, not the entire catchment is modelled because the water authority is mainly interested in the discharge entering at the Dutch-Belgian border. Therefore, the gauge at Hommerich is modelled. The area contributing to the Geul river up to Hommerich is outlined in red in Figure 2.1.

Table 2.1: Land cover of both catchments, based on Corine Land Cover 2018 data [1].

Land cover	Geul	Rur
Urban fabric	14.2%	13.3%
Industrial, commercial and transport units	1.6%	2.9%
Mineral extraction sites	>1%	1.0 %
Dump sites	>1%	>1%
Sport and leisure facilities	1.4%	>1%
Arable land	5.6%	29.9%
Pastures	27.5%	17.5%
Heterogeneous agricultural areas	35.9%	1.0%
Broad-leaved forest	4.2%	10.7%
Coniferous forest	1.0%	16.2%
Mixed forest	8.4%	3.8%
Moors, heathland and natural grasslands	>1%	1.7%
Peat bogs	0	>1%
Water bodies	0	>1%
Total	100.0%	100.0%

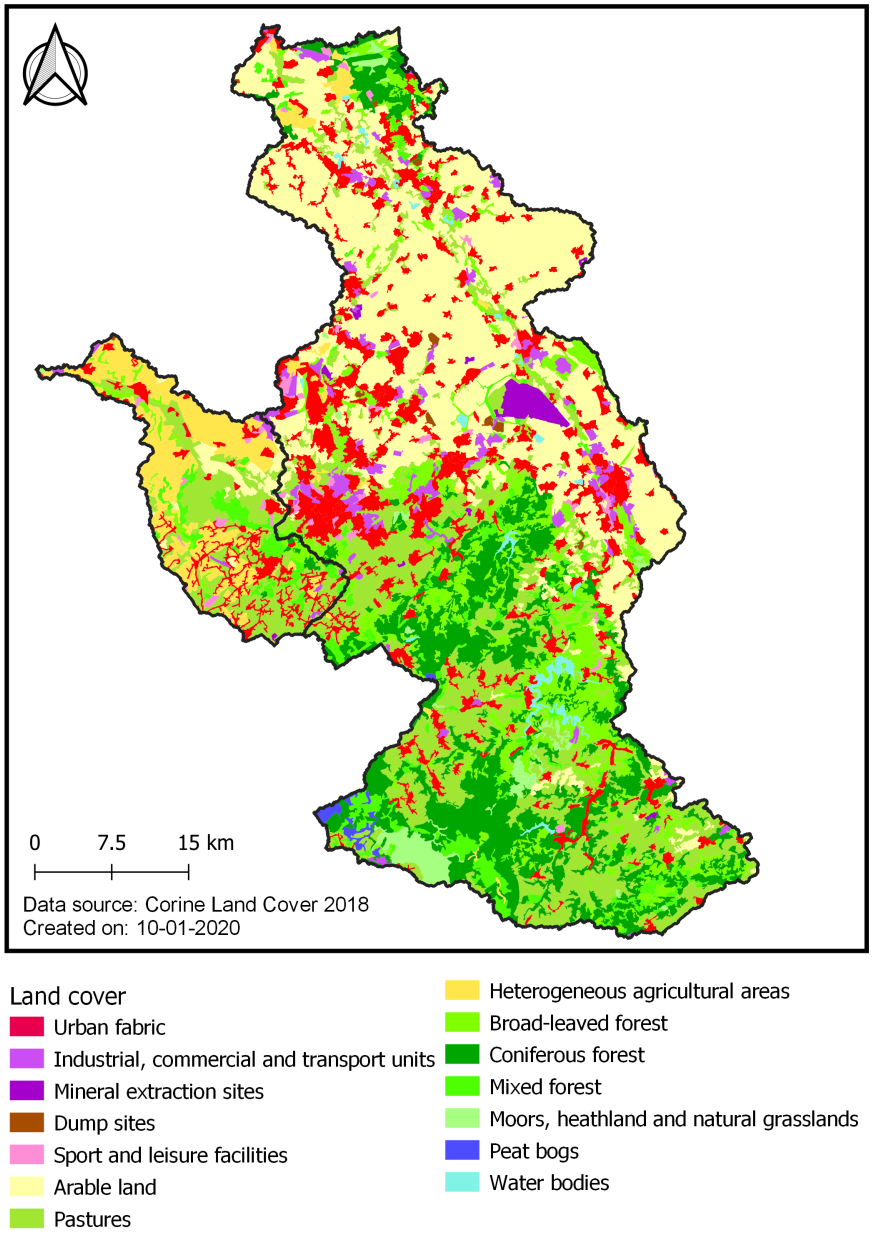

Figure 2.2: Map showing land cover of the Geul and the Rur catchments (separated by the black line). Corine Land Cover 2018 data is used [1].

2.1.2. THE RUR RIVER

The Rur catchment covers about 2360 km^2 and is a subcatchment of the Meuse, covering 7% of the Meuse catchment [63]. The Rur river, in turn, is fed by three smaller rivers; the Wurm, Merzbeek and Inde river. The lion's share of the catchment is situated in Germany (88%), leaving 7% and 5% for Belgium and the Netherlands, respectively. The altitude differs from 15 meters above mean sea level, at the mouth of the Rur, up to 690 meters in the Eifel. An elevation map of the Rur catchment is shown in Figure 2.3. The average river gradient is 0.0039 m/m. The Rur's source is at the High Fens in Belgium. After 165 kilometres it flows into the Meuse at the Dutch city of Roermond (which is Dutch for 'mouth of the Rur').

The Rur catchment is in many ways divided into two parts by a horizontal line at the city of Aken (Aachen in Figure 2.3). Situated below this imaginary line is the mountainous region called 'Eifel'. It consists of Palaezoic and Mesozoi solid rocks. Due to low groundwater storage capacity and permeability, water can only flow along faults. This results in most rainfall quickly transforming to surface run-off [64].

The northern part of the catchment has different physiological conditions, consisting mainly out of the sed-

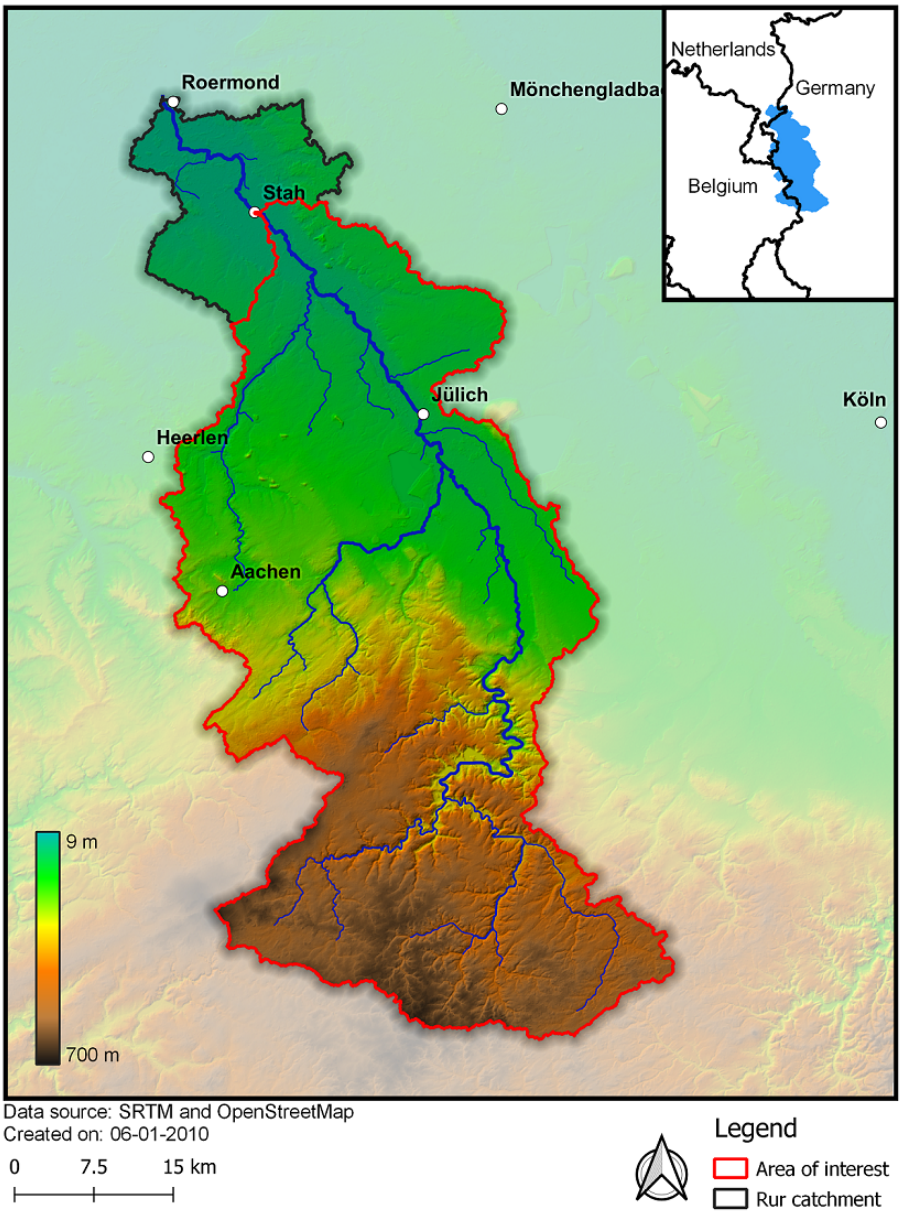

Figure 2.3: Elevation map of the Rur catchment.

iments washed away from the southern part. The near surface rocks consist of eolian deposits that form the fertile top soil layer. The high hydraulic conductivity in this area allows for lowering of the groundwater level for open pit brown coal mining. Although open pit mining covers only 1% of the catchment area, its impact is great because of the reshaping of the area and the lowering of the groundwater table. [65]

About an eight of the catchment surface area is taken up by urban areas, most of which concentrated near rivers. Forests cover roughly a third of the catchment. About half the catchment is arable land or pasture. Peat bogs, moors and heathland are situated in the most southern area of the catchment. Land cover of the catchment is shown in Figure 2.2. Some areas are designated as nature reserves, the main one being the national park Eifel. Most of the Rur river has been subject to river training, save some exceptions. These exceptions are now designated as nature reserves.

The catchment is classified as Cfb under the Köppen climate classification system. Average annual precipitation ranges from 1300 mm in the southern part to 550 mm in the north. Average annual precipitation over the entire catchment is 885 mm/year. Again, the catchment is split when it comes to the timing of the precipitation. In the north most of the rain is concentrated during summer, whereas it is concentrated during winter

for the southern mountainous region. [65]
The low groundwater storage capacity combined with high winter precipitations have caused floods in the past. Peak discharges were estimated at 450 m^3/s. During summer months flow was often less than 1 m^3/s. The floods and large range of discharges over the year resulted in the construction of a series of reservoirs in the southern part. In total nine reservoirs have been constructed between 1905 and 1981 with a total storage volume of 300 million cubic meter. The dams did not just serve for flood control, but also for water supply, low flow elevation and water power generation. Due to this system, peak discharges of the last reservoir has been limited to 60 m^3/s and minimum summer flow has increased to 5 m^3/s. Average flow is 22 m^3/s at Stah. [65]
Leisure activities such as swimming and sailing and tourism have gained importance during the last decades. Ecology is also of increasing importance, resulting in more ecological friendly reservoir operations. [65]

Not the entire basin is modelled, but up to the gauge at Stah. While it would be preferable to include the discharge time series at Jülich as this implicitly contains the reservoir operations in the south, this is difficult. Reason for this is that it takes twelve to fourteen hours for water to flow from Jülich to Stah. Forecasting longer than this period ahead would require forecasts of discharge at Jülich, which would require an additional model. If this lag would be larger, discharge of the area between Jülich and Stah could be modelled and then, with some time lag, added to the discharge at Jülich.

2.2. VARIABLES

Several variables are used in this thesis. In this section it is explained why these variables are chosen, how they are transformed if applicable, and the variables' metadata. The meteorological data used is either obtained from a KNMI station or ERA5 Reanalysis [66].
The Royal Netherlands Meteorological Institute (KNMI) is the Dutch national weather service. Primary tasks are monitoring of weather, climate, air quality and seismic activity and weather forecasting. It is also the national research and information centre for climate, meteorology, air quality, and seismology. They maintain a weather measurement network consisting of 48 automated stations [67]. One of these stations is located near the city of Maastricht, see upper left corner of Figure 2.1.
ERA5 is the latest climate reanalysis done by the European Centre for Medium-range Weather Forecasts (ECMWF). Periodically, ECMWF combines its forecast models and data assimilation system to 'reanalyse' observations. At the end, a global data set is created that estimates hourly atmosphere, land surface and ocean data. The data set is used for research and education, commercial ends and for monitoring climate change. What it basically does, is combining today's state-of-the-art weather models with observations made in the past to deliver a consistent and complete picture of the past weather. In this study the 'ERA5 hourly data on single levels from 1979 to present' is used.
For the Geul catchment, the KNMI data is used. Although the station is situated outside the catchment, it is preferred to ERA5 as the former are actual measurements whereas the latter are estimates. An exception is precipitation. ERA5 precipitation is used for the Geul too as it seems better to use a catchment average compared to a point measurement outside the catchment. In addition, most peak discharges are often caused by convective rainfall events, which will be often missed by a point measurement.
For the Rur catchment, ERA5 reanalysis data is used. There are no measurement stations located in or near the catchment that have consistent time series of the meteorological data. Therefore, satellite data is used.
Data has been collected from October 2008 up to and including September 2018 for the Geul catchment. For the Rur, this is October 2008 up to and including September 2016. The hydrologic year 2017/18 is discarded because it has a missing period of roughly two months. 2016/17 is removed because it does not have any high magnitude flows compared to the other years. In addition, data for the Rur catchment has a daily resolution to reduce the magnitude of grid data required. While more data is normally beneficial for ML models, a period of maximum ten years is chosen to keep processing feasible.
Table 2.2 gives a summary of the variables including resolution and source. Each variable is discussed more in-depth here below.

2.2.1. DISCHARGE

In this study, discharge will be predicted. The data was retrieved from the water authority Waterschap Limburg. Water level is measured every fifteen minutes and then automatically converted to discharge using rating curves.

Table 2.2: Source and resolution of the used variables. NDVI is normalized difference vegetation index, SMC is soil moisture content, and ASML is above mean sea level.

	Geul			Rur		
Variable	source	temporal	spatial	source	temporal	spatial
Discharge	WL	15-minute	point	WL	15-minute	point
NDVI	NOAA	daily	6x6 km	NOAA	daily	6x6 km
SMC	ESA-CCI	daily	25x25 km	ESA-CCI	daily	25x25 km
Temperature	KNMI	hourly	point	ERA5	daily	31x31 km
Wind speed	KNMI	hourly	point	ERA5	daily	31x31 km
Relative humidity	KNMI	hourly	point	ERA5	daily	31x31 km
Solar radiation	KNMI	hourly	point	ERA5	daily	31x31 km
Pressure (AMSL)	KNMI	hourly	point	ERA5	daily	31x31 km
Precipitation	ERA5	hourly	31x31 km	ERA5	daily	31x31 km

For the Geul, data from October 1st 2008 to September 30th 2018 is used. Multiple gauges are available to model, namely Hommerich, Sippenaeken and Cottessen. The latter two are both situated around the Dutch-Belgian border, Cottessen being Dutch and Sippenaeken Belgian. The gauge of Sippenaeken is not owned by WL and there is no operational data transmission to the water authority. At Cottessen discharge is measured using a measurement gutter. While this is more accurate if water level is within the calibrated range, above this range the measured discharge cannot exceed the upper boundary. Therefore, Hommerich is chosen as the station to be modelled.

For the Rur, only two measurement stations are owned by the water authority; at Stah and Jülich. Both time series go back at least 24 years and are measured on a fifteen minute interval. It is decided to model Stah because this location is closest to the Dutch-German border, and thus closer to the area of WL. In addition, WL bases it decisions on the discharge at Stah.

Not all three time series are complete nor validated. Values can be missing because of maintenance, a dysfunction of the water level recorder or an error in transmission to give some examples. Measured values can be wrong due to a non-natural increase or decrease of water level. This was checked manually for the two time series. Outliers, both highs and lows are easy to spot, and can be checked by comparing the point to the same point of another station upstream or downstream. Outliers are corrected by taking the average of the measurement before and after the specific point.

In Table 2.3 some information regarding the three stations is given.

Table 2.3: Total and missing data points for discharge of both rivers for several resolutions.

	Geul (Hommerich)	Rur (Stah)
Period	1-10-2008 - 30-9-2018	1-10-2008 - 30-9-2016
Total data points	350592	280512
Missing (15-minute)	1263 (0.36%)	4253 (1.21%)
Missing (hourly)	249 (0.28%)	975 (1.11%)
Missing (daily)	0 (0%)	37 (1.01%)

2.2.2. PRECIPITATION

Precipitation is a key variable in hydrology, if not the most important one. It is a forcing to rainfall-runoff models. In this study total precipitation from ERA5 Reanalysis is used. Total precipitation is the "accumulated liquid and frozen water, including rain and snow, that falls on the Earth's surface" [66]. It contains both large-scale precipitation and convective precipitation. What it does not include is fog, dew and precipitation that evaporates before it hits the surface. It is given in depth as if the water would spread equally over the grid cell. Its temporal and spatial scale are hourly and 31x31 km, respectively. Again, the precipitation from ERA5 is used for both catchments for the reasons given in Section 2.2.

2.2.3. NORMALIZED DIFFERENCE VEGETATION INDEX

The Normalized Difference Vegetation Index (NDVI) is one of the most widely used indices related to vegetation, and computed using remote sensed data [68]. The formula for NDVI is

$$NDVI = \frac{NIR - vis}{NIR + vis} \tag{2.1}$$

where NIR is the near-infrared radiation and vis the visible radiation. The ratio is always between -1 and +1 [69]. Negative values often represent water. Values around zero indicate barren areas, snow or sand. Increasingly higher values indicate shrub and grassland or temperate and tropical rainforests [70]. Basically, NDVI gives an indication of the density of green in a specific area.
NDVI is used in this study as the amount of vegetation affects run-off. This is especially true for vegetation in and around waterways as vegetation decreases cross-sectional average flow velocity [71]. A lower flow velocity has two effects: it increases the delay time from stream to river and allows more water to infiltrate, reducing the total volume reaching the river. Thus, dense vegetation reduces total river flow and distributes it over a longer period. Given that just over half the catchment is used as either pasture or agricultural land, NDVI can have a significant effect on the discharge.
Data was retrieved from the NOAA National Centers for Environmental Information. It has a spatial resolution of 0.05° by 0.05°, a daily temporal resolution, and is updated daily as well [72].

2.2.4. SOIL MOISTURE CONTENT

The soil moisture content (SMC) is the amount of water in the top layer of the soil, typically up to the first two meters [73]. In mathematical form, the volumetric water content, θ is often used:

$$\theta = \frac{V_w}{V_{wet}} \tag{2.2}$$

where V_w is the volume of water and V_{wet} the sum of total volume of wet material (i.e. soil, water and air). Both variables are expressed in cubic meters. The volumetric water content is always between zero and one. Zero indicating no water present in the soil, and a value close to one indicating extremely wet soil.
In hydrology SMC is an important state variable as it represents the wetness of a catchment prior to rainfall [74, 75]. This is especially true in operational hydrology [54]. It is often the principal factor in determining whether rainfall becomes surface run-off or infiltrates into the ground [55]. The variable is used in this study exactly for that purpose, during wet conditions relatively more precipitation should convert into surface run-off than during dry conditions.
In this study the ESA CCI SM Combined dataset is used [76–78]. This dataset is based on data from the AMI-WS, ASCAT-A and ASCAT-B sensors. It has a spatial resolution of 0.25°and has a daily temporal resolution [79].
Out of a total of 3652 data points, 216 and 215 are missing for the Geul and Rur catchments, respectively. Each ML algorithm has a different way of dealing with missing data. This is discussed in Section 3.3.1 for gradient boosting and in Section 3.4.3 for deep learning.

2.2.5. TEMPERATURE

Temperature is included in this study because of its relation to potential evaporation and soil moisture content. A high temperature often indicates high potential evaporation. The reverse is true for SMC. A high temperature often results in a low SMC, as all water available in the soil is more likely to evaporate. The temperature two meter above ground level is used. For ERA5 its temporal and spatial scale are hourly and 31x31 km, respectively. For KNMI it is a point measurement and measured hourly.

2.2.6. SOLAR RADIATION

Surface net solar radiation is used to calculate the reference evaporation according to the method advocated by STOWA [80]. The following equations are used.

$$ET_{ref} = \frac{0.65 \cdot s \cdot K_{in}}{(s+\gamma) \cdot \lambda \cdot \rho} \tag{2.3}$$

$$\gamma = 0.0646 + 0.00006 \cdot T \tag{2.4}$$

$$\lambda = (2501 - 2.375 \cdot T) \cdot 1000 \tag{2.5}$$

$$e_s = 0.6107 \cdot 10^{\frac{7.5 \cdot T}{237.3+T}} \tag{2.6}$$

$$s = \frac{7.5 \cdot 237.3}{(237.3+T)^2} \cdot ln(10) \cdot e_s \tag{2.7}$$

In the equations above ET_{ref} is the reference evaporation in mm/d, s the slope of the vapour pressure curve in kPa/°C, K_{in} the global incoming radiation in J/(m^{2}h), γ the psychrometric constant in kPa/°C, λ the heat of vaporization in J/kg, ρ the specific weight of water kg/m^3, e_s the saturation vapor pressure in kPa and T the measured hourly temperature in °C.

KNMI measures global incoming radiation on a hourly scale and is a point measurement. From ERA5 the surface net solar radiation is used.

2.2.7. RELATIVE HUMIDITY

Relative humidity is selected for this study because of it interaction with potential evaporation and SMC. The relative humidity from ERA5 Reanalysis is used [66]. Its temporal and spatial scale are hourly and 31x31 km, respectively. It is defined as the water vapour pressure as a fraction of the value at which the air becomes saturated. ERA5 uses three regimes to calculate relative humidity; if temperature is above 0°C, it is calculated for saturation over water, if it is below -23°C as saturation over ice, and between those two temperatures it is interpolated between ice and water values.
KNMI measures relative humidity directly using a hygrometer [81].

2.2.8. WIND SPEED

This variable is included because a high wind speed correlates with precipitation [82]. Physically, higher wind speeds encourage more evaporation, which in turn destabilizes the boundary layer and can lead to convective precipitation.
It is measured at a height of ten meters and its unit is meters per second. It is the average wind speed over the past period in a cell or point. For ERA5 its temporal and spatial scale are hourly and 31x31 km, respectively. For KNMI it is a point measurement and measured hourly.

2.2.9. DATA RESAMPLING

As some data is only available on a daily resolution, and some on a higher resolution, resampling is necessary to obtain the same resolution. Two resolutions are modelled; daily and hourly. In order to go from hourly or higher to daily, data is resampled according to Table 2.4. Downsampling from daily to hourly is simply done by taking the daily value for all hours of that specific day. This is quite a big assumption for soil moisture content as it increases relatively much after a precipitation event. For NDVI this is not as much a problem because it follows a more seasonal pattern. The greenness of a plant does not change much during the day except for crops when harvested.

Table 2.4: The method used for resampling from hourly to daily for each variable. Note that SMC and NDVI are missing as these variables have a daily temporal resolution already. In case of downsampling to hourly, the same daily value for SMC and NDVI is used all hours of that day.

Variable	Method
Discharge	Maximum
Temperature	Mean
Wind speed	Mean
Pressure	Mean
Relative humidity	Mean
Reference Evaporation	Sum
Precipitation	Sum

2.3. OPEN-SOURCE SOFTWARE

For this thesis extensive use of open-source software has been made. The programming language used is Python 3.7 [83]. For data preprocessing and management Scikit-learn [84], Pandas [85], Numpy [86], xarray [87] have been used. Keras [88] and Tensorflow [89] were used to program the LSTM model. XGBoost was used to program the GBDT [3]. The GR4J package was used for to implement the GR4J model [5]. All figures have been made using Matplotlib [90].

3

METHODOLOGY

This chapter consists of five sections. The first section describes forecasting strategies. The second one explains some important machine learning concepts. The third one covers the gradient boosting model, and the fourth section the deep learning model. The final section treats the conceptual model to which the ML models are compared.

3.1. FORECAST STRATEGIES

There are two basic methods to forecast more than one time period ahead; direct and recursive forecasting. When using direct forecasting, a separate model is built for each time horizon, see Equation 3.1. Alternatively, a single model is created to make a one-step forecast, and this model is used recursively to forecast the next step using prior forecasts, see Equation 3.2. This is recursive forecasting [91, 92]. in the equations below, $\hat{y}$ is the predicted value or array of values, y_t the input values or array, M the number of inputs and H the amount of steps ahead.

$$\hat{y}_{t+h} = f_h(y_t, y_{t-1}, ..., y_{t-M+1}) \text{ and } 1 \leq h \leq H \tag{3.1}$$

$$\begin{aligned} \hat{y}_{t+1} &= f_1(y_t, y_{t-1}, ..., y_{t-M+1}) \\ \hat{y}_{t+2} &= f_1(\hat{y}_{t+1}, y_t, y_{t-1}, ..., y_{t-M+2}) \end{aligned} \tag{3.2}$$

In this study direct forecasting is used because this method does not require forecasts of input variables. Also, direct forecasting is based solely on measurements and not on earlier forecasts. In addition, recursive forecasting suffers from accumulation of errors, especially from the point on that only forecasts are used to forecast. A drawback of the direct method is its higher computational demand because as many models as the time horizon have to be trained [93].

3.2. MACHINE LEARNING CONCEPTS

In this section two concepts are discussed. Firstly, the general idea of building a supervised machine learning model and secondly the concept of overfitting. Other concepts are introduced when appropriate but having a notion of how a machine learning model functions makes it easier to understand following texts. In addition, the concept of overfitting is one of the most important ones in the realm of machine learning, and therefore introduced here.

3.2.1. BUILDING A SUPERVISED MODEL

The idea behind a supervised ML model is quite simple. Data has to be fed to the appropriate algorithm, which depends on the task. The algorithm tries to learn the principles governing the training data, when this is finished it is called a trained model. At this point, unseen data can be fed to the model in order to get predictions. A flowchart is shown in Figure 3.1.

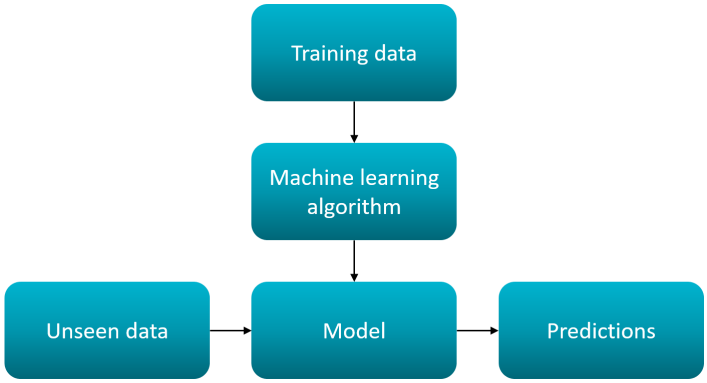

Figure 3.1: The concept of a supervised ML model.

3.2.2. OVERFITTING

Overfitting simply means that a model is learning to reproduce the training data instead of the principles behind it. It is perhaps best explained by Figure 3.2. In terms of training loss the model on the right will be lower compared to the model on the left. However, if both models are applied to a test set, the performance of the latter model will be better than the former. This is because the former model has learnt to replicate the noise too.

Overfitting is a fundamental issue in machine learning and stems from the tension between generalization and optimization. It can be counteracted by implementing regularization techniques, which will be discussed for each ML model separately.

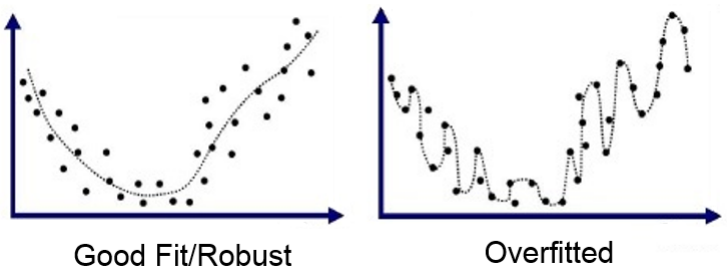

Figure 3.2: Visualization of a underfitted, generalized and overfitted model. Image altered from [2].

3.3. GRADIENT BOOSTING

In this section the gradient boosted model is discussed. It is explained how the model works, what its hyperparameters are and how these are optimized, how training is done and how the ensemble interval is constructed.

Gradient boosting [94] is a machine learning technique for both regression and classification problems. This method provides predictions via a combination of weak learners, typically decision trees. A weak learner performs just slightly better than by random estimate. A model is built by adding new learners and generalizes them by allowing optimization of an arbitrary loss function. It is best explained using an example, e.g. a least-squares regression problem. The goal is to calibrate the model parameters F to predict values $\hat{y} = F(x)$, where x is the input data, by minimizing the mean squared error (Equation 3.3) over the entire training set.

$$\frac{1}{n}\sum_{i=0}^{n}(\hat{y}_i - y_i)^2 \tag{3.3}$$

In the equation above y is the true value of the output variable and n the amount of samples. Now, at each iteration m, except the first tree, there is an imperfect model F_m that can be improved. The algorithm improves this by creating a new tree that adds an estimator h to provide a better model than the last one:

$$F_{m+1} = F_m + h \tag{3.4}$$

In order to find h, the solution starts with the perfect h, which is the difference between the prediction and observed value:

$$F_m + h = y \text{ or } h = y - F_m \tag{3.5}$$

Gradient boosting is fitting h to the residual of the actual value and the latest model prediction. Each iteration attempts to correct the error of the latest model. The model keeps 'growing' trees until the error is close to zero or the decrease in loss is below a certain threshold. If this point is surpassed the algorithm is learning to replicate training data and not the principles behind it, this is where overfitting begins.
For this regression example, as the name indicates, the mean squared error is minimized. In more general terms, a loss function is minimized.

3.3.1. XGBOOST FRAMEWORK

One Python library that is based on the gradient boosting method is XGBoost [3]. It stands for eXtreme Gradient Boosting and is slightly different from ordinary gradient boosting. Besides some minor computational improvements, it minimizes an objective function (Equation 3.6) that consists of a loss function and a regularization function whereas normal boosting only optimizes a loss function.

$$Obj(\theta) = L(\theta) + \Omega(\theta) \tag{3.6}$$

So instead of just measuring how well the model fits on the training data, it also measures complexity of the model. The algorithm tries to finds a trade-off between a good predicting model (L) and to keep it as simple as possible (Ω). A visualization of how this works is given below in Figure 3.3.

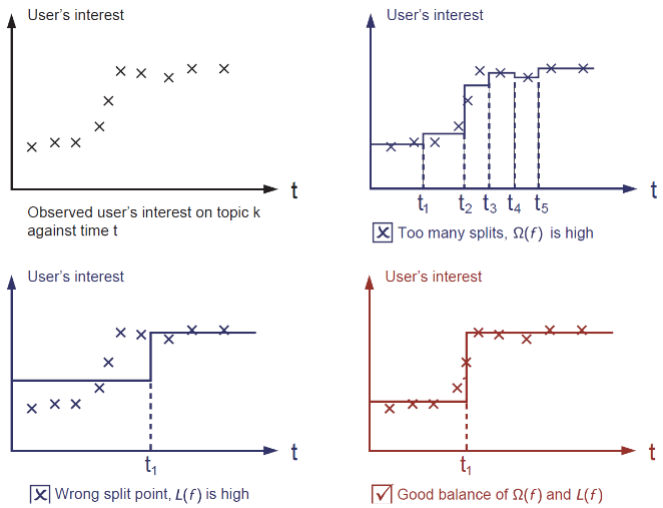

Figure 3.3: Visualisation of how the objective function works. Upper left figure shows the data. Upper right shows a model that is accurate but also complex and might be overfitting as noise is modelled too. Bottom left shows a simple model but with low accuracy. Bottom right shows a simple but accurate model. [3]

Regularization

The concept of regularization is important within the realm of machine learning. It is used to reduce overfitting [95]. There are many regularization techniques available, but in this research dropouts (Section 3.4.4) for the LSTM model and L2 regularization for the GBDT model are used. In both models early stopping and K-fold validation are used.
Early stopping means that the model is trained until performance of the validation set no longer improves, as at this point the model is prone to overfitting. Figure 3.4 shows the concept. There are several implementations as of when exactly to stop. Here, training is stopped if validation set loss has not decreased over 20 epochs (training iterations). If early stopping is triggered, the weights belonging to the epoch that had the lowest validation set loss are chosen. Other implementations can be change of loss below a certain threshold or no change in metric over a given number of epochs.

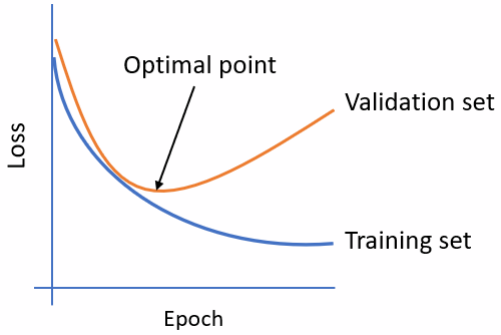

Figure 3.4: The concept of early stopping visualised.

K-fold validation, or cross-validation, means that training, validation and test set are partitioned several times out of the total data set, and the model is trained separately on each partition. At the end, results are averaged. This often slightly decreases model performance compared to a single run but increases generalization and robustness. Figure 3.5 shows how it works.

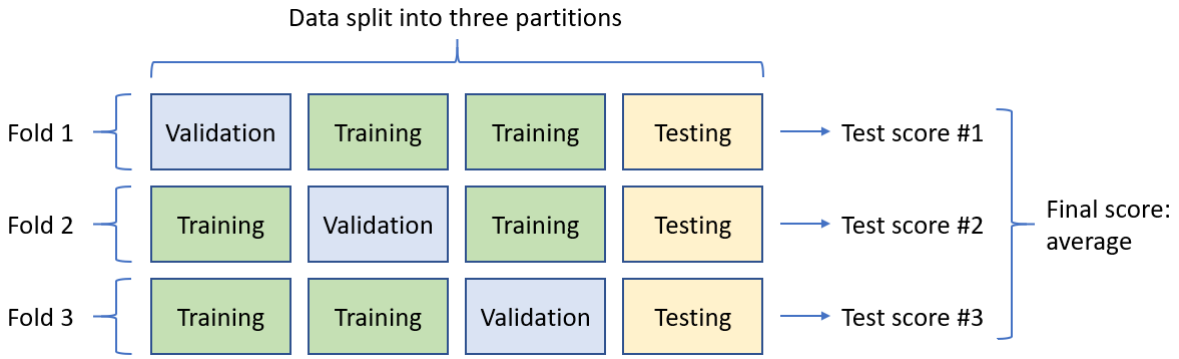

Figure 3.5: The concept of K-fold validation visualised. Image altered from [4].

It is imperative to split the total available data set into three; training, validation and test. The model is trained on the former and validated on the middle one. Once the model is at its best, it is tested on the latter. Having two sets, training and testing, is not enough. Estimating model parameters always includes tuning its configuration (hyperparameters). This tuning is done by using the performance of the model on the validation set as a feedback. Tuning it too little results in the model under-performing. Tuning it too much results in overfitting on the validation set, even though the model is not directly trained on it.
This phenomenon is caused by the so-called information leaks concept. By adjusting a hyperparameter based on the performance of the validation set some information about this set leaks into the model. As tuning often is an iterative process, more and more information leaks into the model. After a while, the model performs exceptionally well on the validation set. Not surprising as the model was optimized for the validation set. However, one cares about the model's performance on unseen data and that is where the test set comes in. The model does not have any information on this set, not even indirectly by tuning. [4]
When splitting the available data, one should check whether all three sets are representative of the available data. If the validation set is, for example, for the most part an unusual dry spell, it is not representative and the validation metric is likely to be an improper representation.

The algorithm of XGBoost is reviewed now. A tree ensemble model uses K additive functions to predict the output $\hat{y}_i$ (Equation 3.7). The prediction is the summation of the outcomes of all trees.

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i) \tag{3.7}$$

In order to learn a given training set an objective function $\mathscr{L}$ is minimized (Equation 3.8). l is an arbitrary loss function. γ is a hyperparameter that controls regularization. The higher γ is, the higher the regularization. T is the amount of leaves in a tree. λ is the L2 regularization (also known as Ridge regression) term on weights. w represents the leaf weights and are the outcomes of a tree. Note that the first and second term on the right hand of Equation 3.8 are the loss and regularization term, respectively. An example of how Ω works is given in Figure 3.6.

$$\mathscr{L} = \sum_i l(\hat{y}_i, y_i) + \sum_{k=1} \Omega(f_k) \tag{3.8}$$

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda ||w||^2 \tag{3.9}$$

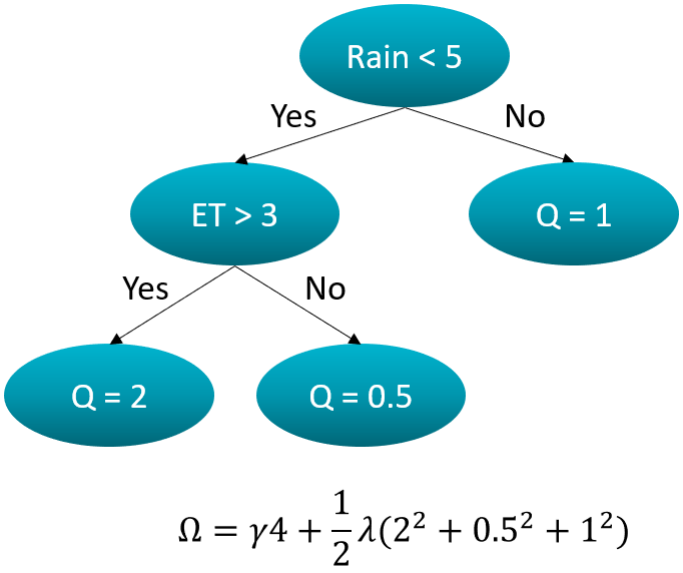

Figure 3.6: An example of how Ω is calculated for a fictive tree.

Equation 3.10 is a combination of Equation 3.5 and 3.8, where the new prediction is equal to the old prediction ($\hat{y}_i^{t-1}$) plus a correction in the form of a new tree (f_t). The goal is to find a new tree that minimizes the objective function below.

$$\mathscr{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \Omega(f_t) \tag{3.10}$$

The next step is to take the derivative of the loss function. In order to reduce complexity, XGBoost calculates the second order Taylor expansion to approximate the loss, see Equation 3.11. Here g_i and h_i are the first and second order of the Taylor expansion of the loss at previous iteration with respect to predictions ar previous iteration, respectively.

$$\begin{aligned} \mathscr{L}^{(t)} &\simeq \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{t-1}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \\ g_i &= \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \\ h_i &= \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \end{aligned} \tag{3.11}$$

After removing the constant loss term, the new objective function at iteration t is given by Equation 3.12.

$$\tilde{\mathscr{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{3.12}$$

Now, say that f_t has T leaf nodes, I_j is the set of samples belonging to node j and w_j is the prediction for that node. Given this, one can write $f_t(x_i) = w_j$ for a sample i belonging to I_j. With this, $f_t(x_i)$ can be substituted

by the respective predictions into Equation 3.12. This results in Equation 3.13.

$$\begin{aligned}\tilde{\mathscr{L}}^{(t)} &= \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i)w_j^2 + \frac{1}{2}\lambda w_j^2] + \gamma T \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T\end{aligned} \tag{3.13}$$

For a fixed tree structure $q(x)$ one can compute the optimal weight w_j^* for each leaf by equating the derivative of the objective function with respect to each leaf node's weight to zero. This results in Equation 3.14.

$$\begin{aligned}\frac{d\tilde{\mathscr{L}}^{(t)}}{dw_j^*} &= 0 \\ \sum_{i\in I_j} g_i + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)\cdot 2\cdot w_j^* &= 0 \\ w_j^* &= -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda}\end{aligned} \tag{3.14}$$

If the optimal weights given by Equation 3.14 are substituted into Equation 3.13, we end up at the best global model loss for adding a fixed tree structure with K nodes, as given by Equation 3.15.

$$\tilde{\mathscr{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T \tag{3.15}$$

So now the algorithm has eased the burden of computing loss using a Taylor function and knows how to find the optimal weights for a fixed tree structure. The last point of concern is how to choose the right structure out of all possible tree structures. XGBoost has a clever way of doing so. It simply starts building a tree and the split that results in the highest loss reduction is chosen. Lets say I_L and I_R are the sample sets of the left and right node after a split, respectively, and I is the union of both sets. Using Equation 3.15 to calculate the loss of the split for each node, the loss reduction, also called gain, after splitting is given by Equation 3.16. The first and second term within the brackets are the scores of the left and right node if split. The third term within brackets is the score of not splitting, and γ is the complexity cost of adding an additional split.

$$\mathscr{L}_{split} = \frac{1}{2}[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I_I} g_i)^2}{\sum_{i\in I_I} h_i + \lambda}] - \gamma \tag{3.16}$$

FINDING THE OPTIMAL SPLIT

With the equation that chooses the best split given, a split finding algorithm calculates the loss for all possible splits on all features. This is known as the exact greedy algorithm. Most current tree boosting implementations use this method. However, it is computationally demanding as it has to go through numerous possible splits for all features. To reduce this demand, XGBoost uses an approximation method. This method first proposes possible splitting points according to percentiles of the feature distributions. Next, the continuous features are mapped into bins using the possible splitting points and statistics are aggregated for each bin. Finally, the best solution is taken from these bins.

Two variants of this approximation method exist; a global one and a local one. The former proposes all possible splitting points during the initial phase of tree construction and keeps using the same splitting points for all levels. The latter re-evaluates the possible points after each split. The local one is better if the model contains deeper trees, the global one if most trees are shallow.

The final piece to the puzzle is proposing possible splitting points. Percentiles of a feature are often used as possible splitting points as the splitting points will then distribute evenly on the data. In addition, as splitting between points that are already well classified is useless, parts of the training set that are not well represented are favoured to be split. This is done by introducing weights. A lower weight indicates low priority to split.

The weighted percentiles are used to find splitting points. The tenth quantile will not be the lowest 10% of the set, but the lowest weighted 10% of the set. A more formal treatment of this method called weighted quantile sketch is given in [3].

HANDLING MISSING DATA

Often some data values are missing in a data set. Samples with missing discharge values are discarded manually before the data is fed to the algorithm. However, XGBoost has a way to deal with missing input values automatically.

Training XGBoost does so by picking a default direction at each split. It is perhaps best explained with an example. Given a simple model that has as input soil moisture content and as output discharge. The model consists of a single tree, as given on the left in Figure 3.7 The samples are given in Table 3.1. The split has been found already, ignoring missing data in the training process using Equation 3.16. Now, the samples with missing values are tried at both sides of the split and a new average total loss is calculated. This loss is lowest for the 'no' branch (6.94 vs 12.61 for 'yes'), see Table 3.1, so this side becomes the default direction. The tree adjusted with missing data is the tree on the right in Figure 3.7. So generally, the algorithm picks the default direction that results in the lowest split loss. The process for picking a default direction is repeated every time a node is split.

Predicting If a required value is missing, the algorithm will choose the default direction. Implicitly, the model assumes a value, or rather a range of values for a missing value when predicting. For example, in Figure 3.7, if the value is missing, it is assumed the value is below 0.19.

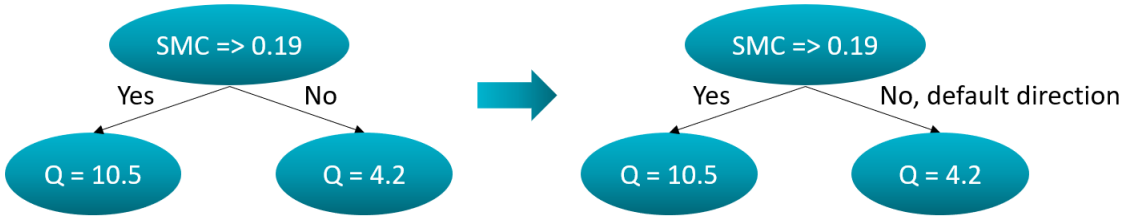

Figure 3.7: The example model. The tree on the left is created ignoring missing data, the tree on the right is corrected for missing data.

Table 3.1: Example data and calculation what the default direction is in XGBoost. Loss function used is root mean squared error. The third column shows the individual contribution of the squared error. An * indicates the default direction.

Discharge	SMC	Loss	Yes*	No	Yes	No*
12	0.35	2.25	2.25		2.25	
10	0.29	0.25	0.25		0.25	
9	0.27	2.25	2.25		2.25	
7	0.16	7.84		7.84		7.84
5	0.12	0.64		0.64		0.64
4	0.07	0.04		0.04		0.04
11	NaN		0.25			46.24
2.5	Nan		64			2.89
4.5	NaN		36			0.09
Average tree loss		2.08	12.61		6.94	

3.3.2. HYPERPARAMETERS & TUNING

All machine learning models have model parameters and hyperparameters. They might seem similar but they have a distinctive difference. Model parameters are internal to the model and its value can be estimated from or trained on the data. These parameters are often saved as part of the trained model and are required by the model to make predictions. Hyperparameters are external to the training process and have to be specified manually [96]. They can (and have to) be tuned to the specific modelling problem as appropriate hyperparameters make the difference between an average model and a state-of-the-art performance [97].

Tuning can be done using an extensive gridsearch, random searching or Bayesian optimization methods [98]. Gridsearch trains the model over and over again until all manually defined sets of configuration have been

tried. While this method may be suitable for small searches, it quickly becomes computationally expensive. Gridsearching a model with five hyperparameters each having ten options results in 10^5 runs.
Random searching is randomly selecting values out of the specified sets. For neural networks, it has been shown empirically and theoretically that this method is more efficient for hyperparameter optimization compared to gridsearch [99]. A drawback of random sampling is that a single poorly chosen hyperparameter range causes the model to perform badly. E.g. if the learning rate range is 0.01-1 but only the range 0.01-0.1 will give good results, 90% of the samples will fail. Therefore, it is key to this method to carefully select appropriate hyperparameter ranges.
Bayesian optimization methods [100] take into account past samples when choosing a new one, resulting in a better estimation for the next run. Even though it is more efficient than gridsearch and random sampling, understanding and implementing the method requires more effort.
However, it turns out that ad hoc manual tuning is a surprisingly effective approach for optimization [101]. The model creator iteratively uses different architectures and hyperparameters to home in on the model's high-performance region. Random searching and coarse gridsearches often provide a good starting point.

XGBoost has plenty of parameters, they can be grouped into general parameters, booster parameters and task parameters.
General parameters consists of the number of parallel threads to run the model on, booster used (e.g. trees or linear functions), verbosity and other options to provide feedback while training. These parameters do not influence the eventual performance but give options to improve it.
Booster parameters are generally known as hyperparameters. There are 20 hyperparameters in total, 7 of which are commonly tuned:

- Learning rate (η); shrinkage of feature weights to make the boosting process more conservative. The lower the learning rate, the more room is given to future trees to correct errors, but more trees are needed.
- Maximum depth of a tree; the deeper a tree, the more complex the model, and the more prone to overfitting.
- Minimum split loss (γ); minimum required loss reduction to make an additional split in a tree. A larger gamma will make the model more conservative.
- Minimum child weight; a node will no longer be split if the sample size belonging to that node goes below a given threshold. This prevents the algorithm of splitting until each sample has a separate leaf. The larger this parameter, the more conservative the model will be.
- Subsample; fraction of all samples that is used to train a specific tree. Lowering this parameter decreases overfitting.
- Column sample by tree; subsample of ratio of columns (features) used when creating a new tree. Reducing the amount of features used per tree reduces the model variance.
- L2 regularization (λ); L2 regularization term on weights, the higher the value, the more conservative the model.

Task parameters specify objective function and evaluation metric. The squared error is used as loss function and root mean square error is used as evaluation metric to compare the training and validation set to check for over- or underfitting [102].

The hyperparameters were tuned by random search. Each model is run 50 times with random hyperparameter values. The hyperparameters of the best iteration are picked as best hyperparameters. More runs are preferable and it is recommended to do so in further studies.

3.3.3. PREPROCCESING & TRAINING

Before training the data is preprocessed. Firstly, missing values are set to 'not a number' (NaN), so XGBoost recognizes the missing values. Next, additional input variables are created by lagging existing ones to create some kind of memory in the model. To give an example, yesterday's rainfall is likely to influence today's discharge. The amount of lag is set using the auto-correlation of each variable, and then tuned to improve the model. With the new features made, samples with missing discharge values are discarded as they are useless for training, validating or testing. If the purpose of the model is to forecast, the discharge values are shifted back by the time horizon, as explained in Section 3.1.
After the dataset has been preprocessed, it is split into two parts; one for training and validation and one for testing. The testing set is always the last year (2017-18 for the Geul, 2015-16 for the Rur) of the available

dataset. The remaining part, roughly 90% of the total data set, is reserved for training and validation. Subsequently, the train and validation set is split again, this time in nine equal parts. With the nine equal parts, each consisting of about a hydrological year, K-fold validation can be implemented. The last year is always reserved for testing. The same XGBoost model is trained on the different partitions of the dataset, as can be seen in Figure 3.5. This results in nine slightly different trained models because the train and validation set slightly differ.
Depending on the dataset, it is not exactly one year per part. As can be seen in Table 2.3, only one discharge time series is complete. This results in each partition consisting of 358 up to 365 days.
For the Rur, the train and validation set is split into seven equal parts.

3.3.4. ENSEMBLE RANGE

As each model gives a deterministic prediction, confidence intervals have to be constructed in an other way. The K-fold validation does not only make the model more robust, it also gives nine, or seven, (slightly) different predictions. Out of these predictions quantiles can be calculated. This method has been implemented for the gradient boosted model. A drawback of this method is due to the models being only slightly different, so are the predictions. This results in a narrow ensemble range.

3.4. DEEP LEARNING

In this section the Long Short-Term Memory model is discussed. It is explained how the model works, what its hyperparameters are and how these are optimized, how training is done and how the confidence interval is constructed. However, before the actual LSTM model is discussed, it is first explained how a neural network works.

The concept of artificial neural networks (ANN) is partly inspired on the nervous system of living beings. The computational model can be thought of as a set of processing units, or artificial neurons, interconnected by artificial synapses. However, at the basis it is a mathematical framework to learn representations from data and to make predictions based on this learnt representations [4].

Artificial neural networks consist out of two components; neurons and synapses. Calculations are done at the neurons, synapses pass on the information. One neuron is taken to explain the main concepts, see figure 3.8. For each neuron, a weighted average is calculated of the inputs x and its respective weights w. To this weighted average a bias b is added. This bias can be thought of as a threshold that must be reached for information to be passed on to the next neuron. Then, the result of that calculation is passed through an activation function and forwarded as input to the next neuron. A neural network consists of three types of layer: input layer, hidden layer and output layer. The input layer is used for the input variables, here information is introduced to the network. An output layer is the last layer of neurons that produces an output (prediction) for the network. A hidden layer is simply a layer between the in- and output layer. A network can have multiple hidden layers.

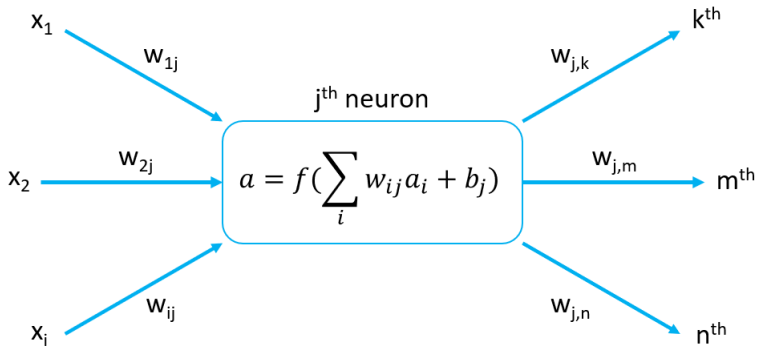

Figure 3.8: A single neuron. The input can either be directly from the input variables or from other neurons. The output is either forwarded to other neurons or is the output of the network.

Activation functions are often non-linear, several functions can be seen in Figure 3.9. Eventually, a prediction will come out at the end of the network. This prediction is then compared with the actual value and the

difference between the two is the loss. The goal of training a model is to minimize the loss, and this is done using backpropagation. Backpropagation means calculating the gradient of the loss with respect to each weight and bias, and then updating the weights and biases. If the loss barely decreases when a weight is changed, the weight remains the same. If the loss decreases when a weight is changed, the weight is changed so that the loss decreases. This process of forward passing and backpropagation is repeated multiple times, each round called an epoch, until the training loss converges. Formally, backpropagation only deals with calculating the gradients and updating the weights is another process, but it is commonly used to denote not just calculating the gradients but also updating the weights.

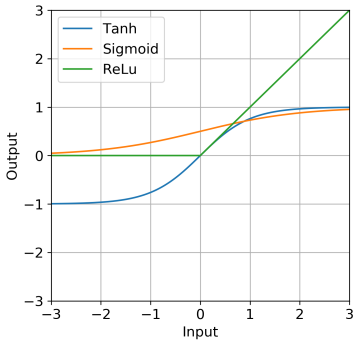

Figure 3.9: Plot of three popular activation functions: hyperbolic tangent (tanh), sigmoid and rectified linear unit (ReLu).

Loss function & Optimization

Every ML model has a loss function that it optimizes. Regression models often use the root mean square error. Classification models often use logistic regression or softmax for binary and multiclass classification, respectively. During the training phase one wants to optimize this loss function, i.e. the error as small as possible without overfitting. Do optimize in an efficient manner there are several learning methods available. They are necessary because finding a minimum in a non-convex loss function is not as easy as one would hope. There are numerous obstacles such as:

- Local minima, if reached some algorithms stop training as it seems like the loss function is optimized.
- Saddle points, where the loss function is constant in most directions, resulting in a near-zero gradient.
- Steep ranges, giving exploding gradients which in turn lead to huge corrections that might nullify earlier training.

For the LSTM model Adam [103] is used. To explain this method it is better to start simple. The most basic method is gradient descent: simply go in the opposite direction of the gradient, or in formula form:

$$L_{i+1} = L_i - \eta \nabla f(L) \tag{3.17}$$

where η is the learning rate, a hyperparameter, that specifies the length of the step between iterations. If it is too large the loss function will oscillate around the minimum, if it is too small many iterations are required and computational demand increases. A disadvantage of this method is that it can get stuck at local minima and on saddle points the gradient will disappear, causing it to get stuck too. Another con is that the algorithm requires the entire use of the training set for each iteration. A way to solve the latter problem is by using mini-batches, dividing the set into smaller batches and training each batch iteratively. This is faster because a smaller amount of data is processed.

The problem of local minima and saddle points is solved by introducing momentum. It does so by taking an exponentially weighted average of past values, thereby reducing noise or local fluctuations. For example, if the algorithm comes across a saddle point it will still continue due to its momentum, the same holds for local minima given that there is sufficient momentum.

In addition to mini-batches and momentum, Adam has a dynamic learning rate instead of a static one. The learning rate decreases faster when the gradient is large, and slower when the gradient is small. This reduces oscillation and will cause the loss function to converge faster, and to mitigates exploding gradients. [104]

In order to demonstrate the training of neural network, a numerical example is given below. This network is made out of an input layer with three nodes, a hidden layer of four nodes and an output layer with one node, see Figure 3.10 on the left. For activation, the sigmoid function is used, given in Equation 3.18. As can be seen in the figures to the right, the weights gradually converge after 45 epochs. Also, the predictions get more accurate over time.

Table 3.2: The four artificial samples used in the model. Discharge is the target variable.

Precipitation	Temperature	Soil moisture content	Discharge
1	12	0.15	1
4	8	0.26	5
2	10	0.22	3
8	6	0.3	8

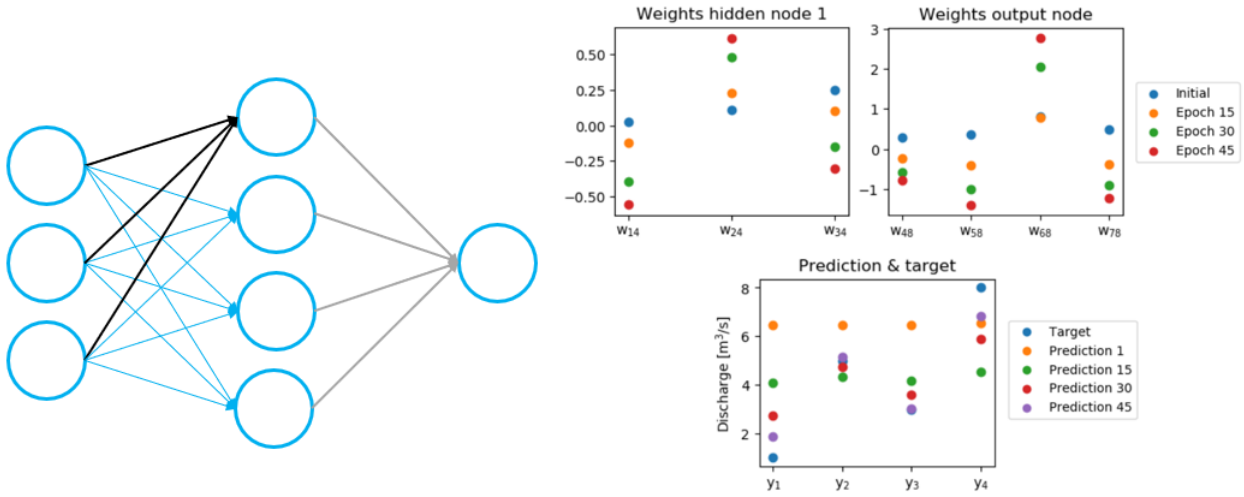

Figure 3.10: Layout of the example. Black arrows point to hidden node 1. Grey arrows point to the output node. It can be seen that the weights converge to a certain value and that the predictions converge to the targets.

$$\sigma = \frac{1}{1+e^{-a}} \tag{3.18}$$

INITIALIZING WEIGHTS

Neural networks are very sensitive to initial conditions. For this reason, the initial weights (w_0) are chosen based on Glorot initialization, also known as Xavier initialization [105]. This method is able to control the variance of the in- and output of the neural network. Background on this method is given in Appendix A.3.

3.4.1. LONG SHORT-TERM MEMORY NETWORKS

A special category of common ANNs that is able to cope with time dependencies are recurrent neural networks (RNN). This ability to cope with time dependencies is big for hydrologic applications as many hydrologic variables depend on antecedent conditions. However, RNNs can be affected by the vanishing gradient problem. During the training phase when backpropagating, the derivatives of connected neutrons are multiplied. Dependent on the activation function used, each derivative can be small. As RNNs have one layer for each time step, this results in the first few layers having a vanishingly small gradient [106]. This impedes training or stops it all together. Similarly, the exploding gradient problem is concerned with gradients that are too large. One type of RNN that takes care of this problem is the Long Short-Term Memory (LSTM) model [107]. Instead of a normal neuron, the LSTM neurons also have a memory cell, input gate, output gate and a forget gate. Gates regulate the flow of information into and out of the cell. The structure of the LSTM neuron

solves the vanishing gradient problem by regulating the flow of information entering and leaving the neuron. Figure 3.11 shows how an LSTM neuron works. Using this figure process of a LSTM will be explained.

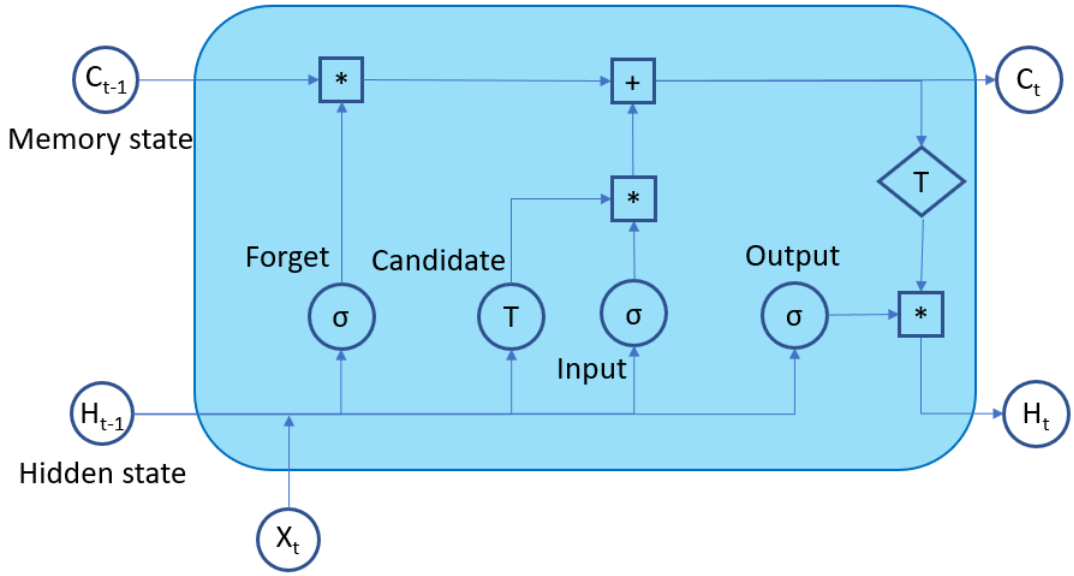

Figure 3.11: Schematization of an LSTM neuron showing how information flows through the gates and alters the hidden and memory state.

The key of a LSTM neuron is the horizontal line at the top. This is the memory state C that remembers past information. Using additional gates, the memory can be altered. The first step is to decide what information of the memory state can be forgotten. This is decided through the forget gate. This gate has as input the previous hidden state H_{t-1} and the input variables X_t, for example temperature and precipitation. The equation for the forget gate is given in Equation 3.19. The sigmoid function is not an arbitrary choice. It is chosen because it filters what information is kept or forgotten. It converts any input value to a number between zero and one. If it is close to zero, information of the previous memory C_{t-1} will be forgotten. If it is close to one, the memory will be kept.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{3.19}$$

The next step is to decide what new information should be stored in the memory state. This is done in two steps; another sigmoid layer decides which values to update (Equation 3.20) and a hyperbolic tangent function creates new candidate values to add to the memory (Equation 3.21).

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{3.20}$$

$$d_t = tanh(W_c x_t + U_c h_{t-1} + b_f) \tag{3.21}$$

Now it is time to update the memory state with the previous outcomes. Before new information is added, old information is forgotten. THe new information to be added is once again weighted by the sigmoid of the input. The update of the memory is given in Equation 3.22. Here the • sign denotes the element-wise product (Hadamard product).

$$C_t = f_t \bullet C_{t-1} + i_t \bullet d_t \tag{3.22}$$

With the memory updated, what is left is to make a prediction which is also equal to the new hidden state H_t. This is done by weighting the hyperbolic tangent of the new memory with the sigmoid of the hidden state and the input variables, see Equation 3.23.

$$h_t = tanh(C_t) \bullet \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{3.23}$$

The memory state keeps track of interdependencies between elements in the input sequence, the input gate regulates to what extent a new value flows into the cell, the forget gate controls how long a value remains in the cell, and the output gate specifies to what extent the value in the memory is used in the activation function of the LSTM neutron.

3.4.2. DATA TRANSFORMATION

Variable ranges often vary greatly from each other. Relative humidity ranges between zero and one, while temperature can range from -10°C to +40° to give an example. For some ML algorithms, variables with a greater range will influence the results to a larger extent, e.g. greater variability in temperature will dominate the lesser variability in relative humidity. This is not desirable as the influence a variable has should not depend on its range. To solve this unwanted tendency, data is often transformed. There are two ways to do this; normalization and standardization. Standardization is only allowed when the data to be transformed is distributed normally. This is not the case for variables in this study, except pressure. Therefore, the data is normalized. This means transforming all variables using Equation 3.24. For the distribution of each variable, the reader is referred to Appendix A.1.

$$X_{scaled} = \frac{X - min(X)}{range(X)} = \frac{X - min(X)}{max(X) - min(X)} \tag{3.24}$$

Where X_{scaled} is the scaled scalar and X the unscaled scalar. This is also known as min-max normalization. [108]

Normalizing data also means losing information about your data. For example, the output range is limited to the output range of the training set. For this reason, extrapolation is not possible using normalized data. While it is not strictly necessary to normalize target variables, it is advisable if the error function is scale sensitive (e.g. least mean squares). If target variables are not normalized, greater targets will be prioritized. Another reason in favour of scaling the target variable is that it allows for more activation functions to be used at the last layer. For example, if the sigmoid activation function is used at the last layer, only values between 0 and 1 can be forecast.

3.4.3. MISSING DATA

Training Neural networks can handle missing data by giving it an arbitrary number that does not have a meaningful value. As data is often transformed to the range 0 to 1, it can be set to any number outside that range. The network will learn that the arbitrary value means 'missing data' and will start to ignore the value. This is simply learnt from exposure to the data. One has to be careful with missing data in the test set. If there are only missing data in the test set, the network has not been able to learn what the missing value is and it will be misinterpret. This can be solved by generating artificial training samples with missing values. [4]

Predicting A neural network cannot make predictions when an input value is missing. Doing so will return a prediction being NaN (Not a Number). The best option is to set missing values to the same value as missing values in the training set. Note that this will degrade performance. [4]

3.4.4. DROPOUT

As mentioned earlier dropout is a regularization technique. Dropouts do so by adding artificial noise which mitigates overfitting [109]. The idea is to randomly drop some nodes each epoch. This can be in the input, hidden and output layer given that each layer consists of more than one node. Because some connections between nodes are lost, the weights will be larger than with all connections present. To nullify this, the weights are scaled using the same dropout probability of each respective layer. For example, if a layer has 20% dropout probability, weights of that layer will be scaled by dividing the weights by 1.20. This happens between training and predicting. While dropouts reduce overfitting, it does increase training time. The major cause for this is the parameter updates being noisy [110]. A visualisation of how dropout works is given in Figure 3.12.

3.4.5. HYPERPARAMETERS & TUNING

A LSTM model has plenty of parameters that can be set manually, see the list below. The optimization is geared towards increasing performance and decreasing computational demand.

- Activation function; the activation function converts the input to output of a cell. Three examples are given in Figure 3.9. Some functions are tailored to classification or probabilities, such as the Softmax function. For thresholds, the binary step function can be used [111]. However, in this study the sigmoid and rectified linear unit (ReLu) are tried.
- Learning rate (η); the learning rate decides in what order weights are updated. If it is low, model performance will increase. However, computational demand increases too as more epochs are required for the weights to converge. Fortunately, Adam has a dynamic learning rate that adapts to the magnitude of values. Relatively large weights will train faster, relative low weights will train slower.

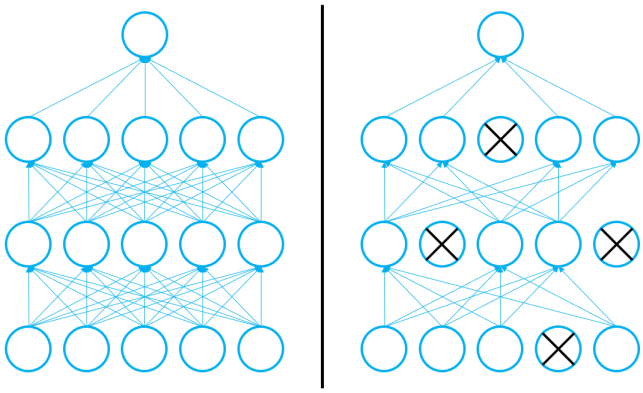

Figure 3.12: Visualisation of how dropout works. A normal neural network on the left and the same network after applying dropout with a 20% of deactivating a node.

- Dropout probability; the percentage that a neuron will drop out. Typical probabilities of losing a node are 0.2 to 0.5 [110].
- Hidden layers width; defines how many hidden layers one should use. Generally, two or three hidden layers are sufficient but there are no clear rules on network width. It is best to start shallow and add a layer until performance no longer increases. [112]
- Batch size; batch size decides how often weights are updated. Training the network on all samples at once makes the training extremely slow. Training the network in smaller batches decreases the training time while keeping performance the same. Typical batch size values are 32 to 128. [113]
- Input window; amount of samples that are fed to the LSTM as one input sample. Or put differently, the number of time steps considered when predicting the output. The window is typically decided on by looking at the partial autocorrelation function of the output variable.

However, Reimers and Gurevych [98] found that the optimizer and dropout have the highest impact for sequence labelling tasks. In addition, Greff et al. [114] found that learning rate can be tuned first using a rather small network before tuning the width and depth of the network. This saves a lot of computational time.
Even though there are more sophisticated methods available to tune hyperparameters, as discussed in Section 3.3.2, optimization is done manually. Gridsearch and Bayesian optimization require a lot of computational resources which simply are not available, especially for the hourly resolution. Three folds are used instead of nine to further reduce computational time, but still have some cross-fold validation. One recommendation is to do this in a more exhaustive way to optimize performance. Each model is ran twice to reduce the influence of randomly initializing the weights from a given distribution, the best performing model is used. More runs would reduce this influence even more, but would also increase total tuning time.

3.4.6. PREPROCESSING & TRAINING

Before training the data is preprocessed according to the following:

1. If the purpose of the model is to forecast, the discharge values are shifted back by the time horizon, as explained in Section 3.1.
2. Data is scaled as discussed in previous section.
3. Missing variable values are set to -1.
4. Data is converted to training samples which have the shape of amount of variables x number of time steps.
5. Samples that have missing discharge values are removed.
6. The remaining samples are split into two parts; training and testing. The former is again split into nine different sets for K-fold validation. The sets are approximately one year each, please refer to Section 3.3.3 as of why this is.
7. For each training and validation combination a separate model is trained and predictions are made. After this, the weights are again randomly initialized.
8. The final prediction is the median of the set of all predictions.

Mean squared error is used for both training loss and evaluation metric. The optimizer used is Adam, as explained in the intermezzo above.

3.4.7. ENSEMBLE RANGE

Nearly all ANNs produce a scalar output. The only exception are deep belief or deep Bayesian networks which give distributions. However, an interval is produced based on the theory of Gal and Zoubin [115] and pseudo code of Klaas [116]. The idea is to use dropouts not just during the training phase but also during forecasting. By randomly removing nodes during the forecasting phase, each time-series will be forecast using a slightly different model. This is numerous times more efficient than training an LSTM over and over again. From this range different forecasts quantiles can be calculated.

3.5. GR4J CONCEPTUAL MODEL

In addition to the two ML models, a conceptual model is implemented. This enables comparison of the ML models with an existing model as benchmark. In conceptual hydrological model, the catchment is schematized as multiple reservoirs storing and exchanging water [117].

The GR4J model (modèle du Génie Rural à 4 paramètres Journalier) is a daily lumped rainfall-runoff model, belonging to the class of soil moisture accounting models [5]. A schematic of the model is given in Figure 3.13. The model simulates daily discharge in millimetre per day, requiring only precipitation and potential evaporation as input. The model consists of two reservoirs; a production store and a routing store. The former one acts as soil reservoir, interacting with rainfall, percolation and evaporation. The latter one accounts for the amount of water that can be stored in the deeper porous soil [118]. In addition, the model allows for groundwater exchange between surrounding catchments [5]. Only four parameters have to be calibrated:

- x_1; maximum capacity of the production store (mm)
- x_2; catchment groundwater exchange coefficient (mm)
- x_3; one-day-ahead maximum capacity of the routing reservoir (mm)
- x_4; time base of the unit hydrograph UH1 (days)

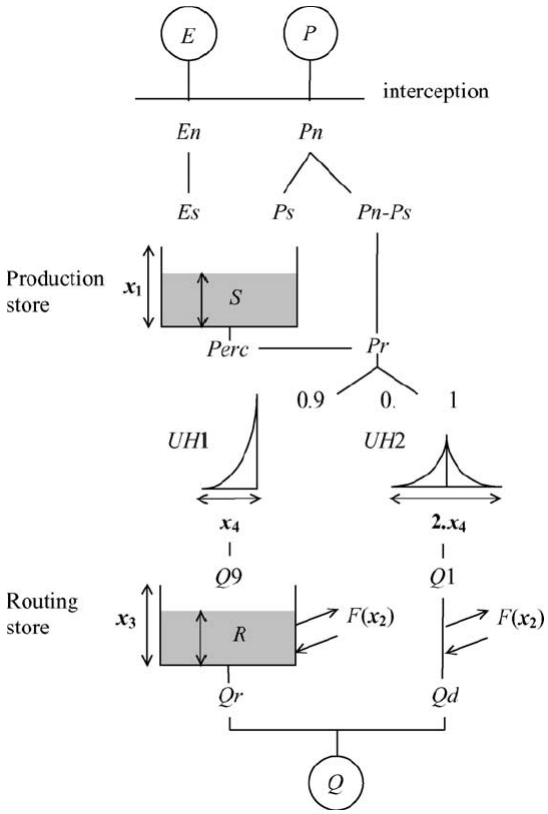

Figure 3.13: Overview of the GR4J model showing the input, output, processes and reservoirs [5].

In this study the GR4J model is selected as benchmark. It is a simple model that has only four calibrated parameters. Still, it has shown good performance in many studies [119]. In addition, GR4J simulates ground-

water flows quite well, which is important as base flow in the Geul is mainly contributed to groundwater [59].

The routine of the model starts with determining whether there is net rainfall or net evaporation using Equation 3.25, where P and E are precipitation and potential evaporation, respectively. The subscript n refers to net amount. Next, if there is a net rainfall, some will infiltrate and increase the soil storage level S if there is capacity left. This infiltration P_s is calculated using Equation 3.26. On the other hand, if there is a non-zero net evaporation capacity E_n, an actual evaporation rate is determined as a function of the soil storage, see Equation 3.27. Now the soil storage can be updated using Equation 3.28. Note that the storage can never exceed the capacity x_1.

$$\begin{aligned} P \geq E, P_n = P - E \text{ and } E_n = 0 \\ P < E, P_n = 0 \text{ and } E_n = E - P \end{aligned} \tag{3.25}$$

$$P_s = \frac{x_1(1-(\frac{S}{x_1})^2)\tanh(\frac{P_n}{x_1})}{1+\frac{S}{x_1}\tanh(\frac{P_n}{x_1})} \tag{3.26}$$

$$E_s = \frac{S(2-\frac{S}{x_1})\tanh(\frac{E_n}{x_1})}{1+(1-\frac{S}{x_1})\tanh(\frac{E_n}{x_1})} \tag{3.27}$$

$$S = S - E_s + P_s \tag{3.28}$$

The percolation is described by Equation 3.29. By definition, percolation is always lower than the storage. Subtracting the percolation from the storage gives the new storage.

$$\text{Perc} = S(1-[1+(\frac{4S}{9x_1})^4]^{-1/4}) \tag{3.29}$$

$$S = S - \text{Perc} \tag{3.30}$$

Next, the total quantity of water P_r that reaches the routing functions is given by Equation 3.31, which is the sum of the percolation and net rainfall minus the infiltration. P_r is split in a fast component and a slow component.

$$P_r = \text{Perc} + P_n - P_s \tag{3.31}$$

The slow component gets 90% of the water and is router first through the unit hydrograph $UH1$ before it is routed through the non-linear routing reservoir. The remaining 10% is routed by the other single unit hydrograph $UH2$. By using two separate unit hydrographs, one can simulate the time lag between precipitation and resulting discharge peak. Both $UH1$ and $UH2$ depend on x_4, the former with a time base of x_4 and the latter with a time base of $2x_4$. Unit hydrograph ordinates are derived from the cumulative proportion of the input with time (also known as S-curves), denoted by $SH1$ (Equation 3.32) and $SH2$ (Equation 3.33).

$$\begin{aligned} t \leq 0, SH1(t) = 0 \\ 0 < t < x_4, SH1(t) = (\frac{t}{x_4})^{5/2} \\ t \geq x_4, SH1(t) = 1 \end{aligned} \tag{3.32}$$

$$\begin{aligned} t \leq 0, SH2(t) = 0 \\ 0 < t < x_4, SH2(t) = \frac{1}{2}(\frac{t}{x_4})^{5/2} \\ x_4 < t \geq 2x_4, SH2(t) = 1 - \frac{1}{2}(2-\frac{t}{x_4})^{5/2} \\ t \geq 2x_4, SH2(t) = 1 \end{aligned} \tag{3.33}$$

With both S-curves known, the unit hydrograph ordinates can be calculated following 3.34, where j is an integer, representing the time in days after a precipitation event.

$$\begin{aligned} UH1(j) = SH1(j) - SH1(j-1) \\ UH2(j) = SH2(j) - SH2(j-1) \end{aligned} \tag{3.34}$$

The next process in the model flowchart is the catchment groundwater exchange F. It affects both the slow and fast routing component. It can be calculated using Equation 3.35. R is the level in the routing reservoir, x_3 its capacity, and x_2 the water exchange coefficient with surrounding catchments. F is physically constrained by x_2, as this is the maximum flux that can be exchanged in one time step.

$$F = x_2(\frac{R}{x_3})^{7/2} \tag{3.35}$$

With the exchange known, the routing reservoir level can be updated using Equation 3.36. The new level is either zero or the sum of the previous level, water exchange and slow routing component. The outflow Q_r is given by Equation 3.37, increasing with higher level in the reservoir. Again, the level is updated. This time by subtracting the outflow (Equation 3.38).

$$R = \max(0, R + Q9 + F) \tag{3.36}$$

$$\text{Perc} = R(1 - [1 + (\frac{R}{x_3})^4]^{-1/4}) \tag{3.37}$$

$$R = R - Q_r \tag{3.38}$$

Lastly, total predicted discharge is calculated by summing the slow (Q_r) and fast (Q_d) component. The latter one is the maximum of zero or the unit hydrograph ordinate of $UH2$ plus the water exchange, see Equation 3.39. The last equation gives the output discharge of the model. [5]

$$Q_d = \max(0, Q1 + F) \tag{3.39}$$

$$Q = Q_r + Q_d \tag{3.40}$$

3.5.1. CALIBRATION

Nine years worth of data, October 2008 until and including September 2017, is used to calibrate the model's parameters. The first year of this period is reserved for warm-up. This is an adjustment process for the model to reach an 'optimal' state. During the warm-up period, the volumes of the reservoirs move from estimated initial conditions to representative values. [120] The initial conditions used here are that both reservoirs are empty at the beginning. The warm-up period is not included when evaluating the performance of the calibration period.
The test year is from October 2017 until and including September 2018. The model is calibrated using the differential evolution algorithm. It is a global optimization algorithm. This algorithm requires two inputs: the objective function to be minimized and the bounds for each model parameter. The former is mean squared error, and the latter is given in Table 3.3. These values are taken from the original paper by Perrin et al. [5]. They indicate approximate 80% confidence intervals for the four parameters. For x_3, the lower bound is 20, but zero is used here based on early calibration tests.

Table 3.3: The bounds used for each parameter for calibration.

Parameter	Lower bound	Upper bound
x_1	100	1200
x_2	-5	3
x_3	0	300
x_4	1.1	2.9

3.6. EVALUATION METRICS

Two metrics are used to evaluate the performance of the models; Nash-Sutcliffe efficency and mean absolute error.
Nash-Sutcliffe efficiency (NSE) is a widely used metric to evaluate the performance of hydrologic models [121]. The formula is given in Equation 3.41.

$$NSE = 1 - \frac{\sum_{i=1}^{T}(Q_m^t - Q_o^t)^2}{\sum_{i=1}^{T}(Q_o^t - \overline{Q_o})^2} \tag{3.41}$$

where $\overline{Q_o}$ is the mean of the observed discharge over all time steps T, Q_m^t the modelled discharge at time step t, and Q_o^t the observed discharge at time step t [122]. It can also be interpreted as the ratio of the mean square error over the observed variance subtracted from 1 [123].
Nash-Sutcliffe efficiency can range from -∞ to 1. If NSE is 0, the observed mean is as accurate a predictor as the model. A negative value means that the observed mean is a better predictor than the model. A value of 1 means that the modelled discharge matches the observed discharge perfectly. Due to the difference between observation and model being squared, this metric is sensitive to outliers and/or high discharges. Because of this tendency, another metric is used that does not favour high or low flows which is the mean absolute error. Moriasi et al. [6] recommended the following performance ratings for monthly time steps (Table 3.4):

Table 3.4: Performance rating according to Moriasi et al. [6].

Performance rating	NSE
Very good	$0.75 < NSE \leq 1.00$
Good	$0.65 < NSE \leq 0.75$
Satisfactory	$0.50 < NSE \leq 0.65$
Unsatisfactory	$NSE \leq 0.50$

Mean absolute error (MAE) indicates the average of the absolute error between the modelled and observed discharge, see Equation 3.42. It is by definition non-negative and has no upper bound. A perfect model would have a MAE of zero, an increasingly high value corresponds with an increasingly worse model. The metric is not weighted towards high or low discharges and is indifferent to the direction of the error [124].

$$MAE = \frac{\sum_{i=1}^{T}|Q_m^t - Q_o^t|}{T} \tag{3.42}$$

3.7. FINDING THE BEST INPUT VARIABLES

Before all simulation can be ran, it first has to be decided which input variables are relevant and lead to the best model performance. This is done for both catchment separately. The procedure consists of starting with a baseline and then trying to outperform this baseline by testing variables. The baseline for both catchment is precipitation and the moving cumulative average of precipitation. The former because precipitation is the main forcing of most rainfall-runoff models, and the latter to account for the volume of water present in the catchment.

1. Select baseline and note down performance of this model
2. Add a new variable, note down performance. Repeat this process until all variables have been tested.
3. Select the variable that increased performance the most. This is the new baseline.
4. Repeat step two and three until performance no longer increases.
5. With the input variables found, repeat step two and three but now by lagging an input variable up to three days.

Some assumptions are made to shorten this process. To start with, it is only done for each catchment using gradient boosting models as these run faster than deep learning models. It is assumed that the final input variables found using the former method are also the best variables for the latter method. Secondly, this procedure is done using daily discharge and it is assumed that found variables are also the best combination for the hourly resolution, and for water depth.
One could simply add all variables and trust on the ML model to only use the appropriate variables, but regarding model parsimony it is better to select only the best performing input variables. Just an increase in model performance when adding an extra variable is not enough to justify an increase in model complexity.

4

RESULTS & DISCUSSIONS

In this chapter the results of both ML models are given and discussed. Firstly, it is shown how the best model variables were selected. Next, the results of the GR4J model are given. Afterwards, the ML models are compared to the conceptual model to see which type performs better in this study. Only the best performing models are shown here. For results of all other models that led to the final models discussed here, please refer to Appendix B.

4.1. RELEVANT VARIABLES

The results of finding the best input variables for each catchment are discussed here. The results for the Geul catchment are given in Table 4.1. From this table it can be seen that the best performing model has as input precipitation, moving five-day cumulative of precipitation, temperature, SMC, and potential evaporation up to and including two days ago (total daily potential evaporation of yesterday and the day before yesterday). While adding wind speed, relative humidity, pressure or NDVI increases performance, the variables in the base model increased performance to a larger extent. For example, adding NDVI to the first baseline increases NSE from 0.44 to 0.49, and decreases MAE from 1.15 to 0.88 m^3/s. Thus, the first result is that wind speed, relative humidity, pressure and NDVI only have a minor effect on performance for the Geul catchment. The increase in model performance when adding an extra variable is not enough to justify an increase in model complexity, only the variables that increase performance the most are used.

The results of finding the best input variables for the Rur catchment are given in Table 4.2. The best performing model has as input precipitation, it's five-day moving cumulative, temperature and SMC up to and including two days ago. What is peculiar is that lagged SMC is important for the Rur. Soil moisture content is a state variable that has its history represented in the current value. To give an example, if it rained yesterday SMC is likely to be high. Given this, it is strange that explicitly accounting for this history of up to two days ago increases model performance. Further analysis is needed to verify whether this is just due to a strong correlation. Note that the difference with the Geul catchment is evaporation, which is not used as input for the Rur. In addition, the NSE of the best performing model of the Rur (0.51) is lower compared to the Geul (0.71). This can be expected as there are reservoir operations in the Rur catchment which are not explicitly accounted for by the input variables.
The hyperparameters of each model given in Table 4.1 and 4.2 are given in Appendix B.1.

Table 4.1: Performance of all models simulating today's discharge of the Geul. The best performing model(s) of each iteration have been marked green. The best overall model is marked bold. The following abbreviations are used: MC: moving cumulative, T: temperature, P: pressure (AMSL), RH: relative humidity, ET: potential evaporation, SMC: soil moisture content, NDVI: normalized difference vegetation index. The subscript $_{t\text{-}x}$ means that a variable is used up to and including x days ago. $ET_{t\text{-}2}$ means that potential evaporation of today, yesterday and the day before yesterday is used.

	NSE			MAE		
Model	train	val	test	train	val	test
Baseline 1: rain, rainMC	0.39	0.23	0.44	1.09	1.15	1.15
Baseline 1 + T	0.53	0.33	0.59	0.94	1.04	0.88
Baseline 1 + WS	0.47	0.28	0.5	1.05	1.13	1.1
Baseline 1 + P	0.52	0.25	0.48	1.00	1.12	1.08
Baseline 1 + RH	0.47	0.28	0.5	1.03	1.11	1.03
Baseline 1 + ET	0.63	0.32	0.58	0.90	1.07	0.96
Baseline 1 + SMC	0.70	0.43	0.59	0.81	0.98	0.88
Baseline 1 + NDVI	0.49	0.29	0.49	1.01	1.10	1.06
Baseline 2: rain, rainMC, T, SMC	0.68	0.46	0.66	0.8	0.94	0.79
Baseline 2 + WS	0.74	0.44	0.64	0.75	0.93	0.80
Baseline 2 + P	0.69	0.47	0.61	0.8	0.93	0.80
Baseline 2 + RH	0.73	0.44	0.66	0.75	0.94	0.78
Baseline 2 + ET	0.75	0.45	0.70	0.75	0.93	0.77
Baseline 2 + NDVI	0.68	0.45	0.63	0.80	0.93	0.80
Baseline 3: rain, rainMC, T, SMC, ET	0.74	0.45	0.7	0.75	0.93	0.77
Baseline 3 + WS	0.84	0.44	0.65	0.61	0.92	0.80
Baseline 3 + P	0.83	0.46	0.63	0.65	0.94	0.82
Baseline 3 + RH	0.81	0.43	0.7	0.67	0.92	0.77
Baseline 3 + NDVI	0.81	0.44	0.66	0.66	0.93	0.79
Baseline 3 + $T_{t\text{-}1}$	0.85	0.44	0.68	0.59	0.92	0.75
Baseline 3 + $T_{t\text{-}2}$	0.77	0.45	0.67	0.73	0.93	0.75
Baseline 3 + $T_{t\text{-}3}$	0.74	0.47	0.68	0.75	0.91	0.79
Baseline 3 + $SMC_{t\text{-}1}$	0.82	0.44	0.69	0.66	0.92	0.77
Baseline 3 + $SMC_{t\text{-}2}$	0.72	0.45	0.69	0.77	0.93	0.77
Baseline 3 + $SMC_{t\text{-}3}$	0.73	0.45	0.67	0.76	0.93	0.78
Baseline 3 + $ET_{t\text{-}1}$	0.82	0.45	0.70	0.66	0.92	0.76
Baseline 3 + $ET_{t\text{-}2}$	**0.84**	**0.46**	**0.71**	**0.64**	**0.92**	**0.75**
Baseline 3 + $ET_{t\text{-}3}$	0.89	0.44	0.71	0.54	0.93	0.75

Table 4.2: Performance of all models simulating today's discharge of the Rur. The best performing model(s) of each iteration have been marked green. The best overall model is marked bold. The following abbreviations are used: MC: moving cumulative, T: temperature, P: pressure (AMSL), RH: relative humidity, ET: potential evaporation, SMC: soil moisture content, NDVI: normalized difference vegetation index. The subscript $_{t\text{-}x}$ means that a variable is used up to and including x days ago. $ET_{t\text{-}2}$ means that potential evaporation of today, yesterday and the day before yesterday is used.

	NSE			MAE		
Model	train	val	test	train	val	test
Baseline 1: rain, rainMC	0.30	0.12	0.24	7.03	7.63	7.37
Baseline 1 + T	0.46	0.24	0.31	6.03	6.88	6.70
Baseline 1 + WS	0.38	0.16	0.27	6.77	7.52	7.36
Baseline 1 + P	0.37	0.17	0.29	6.70	7.33	7.12
Baseline 1 + RH	0.41	0.20	0.27	6.29	6.99	7.48
Baseline 1 + ET	0.46	0.26	0.28	6.13	6.81	6.98
Baseline 1 + SMC	0.43	0.30	0.43	6.07	6.58	6.10
Baseline 1 + NDVI	0.44	0.19	0.26	6.36	7.16	7.24
Baseline 2: rain, rainMC, SMC	0.43	0.30	0.43	6.07	6.58	6.1
Baseline 2 + T	0.52	0.32	0.48	5.60	6.49	5.89
Baseline 2 + WS	0.59	0.31	0.44	5.26	6.54	6.28
Baseline 2 + P	0.52	0.34	0.47	5.63	6.37	5.95
Baseline 2 + RH	0.69	0.29	0.40	4.64	6.58	6.51
Baseline 2 + ET	0.53	0.3	0.4	5.6	6.42	6.38
Baseline 2 + NDVI	0.51	0.32	0.44	5.72	6.46	6.21
Baseline 3: rain, rainMC, SMC, T	0.52	0.32	0.48	5.6	6.49	5.89
Baseline 3 + WS	0.54	0.33	0.47	5.52	6.48	5.91
Baseline 3 + P	0.61	0.32	0.46	5.09	6.40	5.92
Baseline 3 + RH	0.55	0.32	0.42	5.49	6.47	6.36
Baseline 3 + ET	0.57	0.31	0.43	5.36	6.40	6.10
Baseline 3 + NDVI	0.55	0.33	0.48	5.51	6.43	5.93
Baseline 3 + $T_{t\text{-}1}$	0.52	0.32	0.48	5.58	6.44	6.08
Baseline 3 + $T_{t\text{-}2}$	0.55	0.33	0.48	5.49	6.46	5.83
Baseline 3 + $T_{t\text{-}3}$	0.57	0.33	0.49	5.39	6.44	5.81
Baseline 3 + $SMC_{t\text{-}1}$	0.63	0.31	0.49	4.98	6.40	5.81
Baseline 3 + $SMC_{t\text{-}2}$	**0.54**	**0.33**	**0.51**	**5.50**	**6.35**	**5.63**
Baseline 3 + $SMC_{t\text{-}3}$	0.68	0.32	0.50	4.68	6.42	5.81

4.2. GRADIENT BOOSTING

The results of the gradient boosting model, implemented using XGBoost, are discussed here. Results for discharge on a daily are shown for the Geul and the Rur. For the Geul, the logarithmic transformed discharge, water level and hourly discharge are shown too. Each catchment is treated separately. Before the forecast results are shown, discharge based on that same day's variables, so simulations, is discussed first. This because if a model cannot predict today's discharge with today's data, it will not be able to forecast tomorrow's discharge.

4.2.1. GEUL

Before each model is discussed in depth a comment is given about one observation in the dataset. A hydrograph is shown in Figure 4.1. At day 215 (30-04-2018) measured discharge is 23.1 m^3/s but all models fail to simulate this peak, predicting at most 3-5 m^3/s. On this day, rain is not extraordinarily high (10 mm), neither is the moving cumulative of rain (24 mm). NDVI and SMC are 0.22 and 0.32, respectively, indicating favourable circumstances for high discharge as the soil is wet and there is not much vegetation. However, even with perfect conditions there needs to be rain in order to increase discharge. This might indicate a precipitation data error and not a model error. Therefore, this point is dropped from the test set for all Geul models.

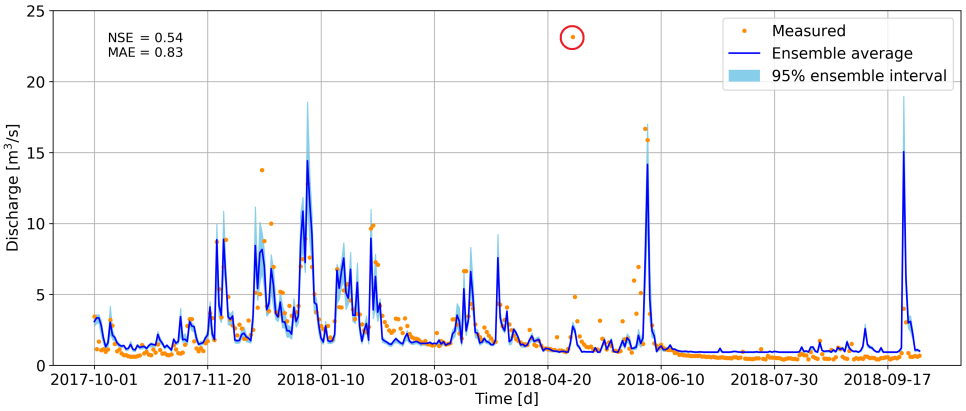

Figure 4.1: Hydrograph including the data error point encircled in red.

DAILY DISCHARGE

The left of Figure 4.2 shows a scatterplot of the predicted and measured discharge, including a dashed 1:1 line. Ideally, all points lie on the dashed line as this indicates predicted discharge is equal to measured discharge. If a point is above (below) this line, the discharge is overestimated (underestimated). The right of Figure 4.2 shows the NSE for each fold.

What can be seen in the scatterplot is that most points are fairly close to the dashed line. As discharge increases, the distance to the dashed line increases too. This is not strange as it is difficult to accurately model. Figure 4.3 shows the hydrograph of the measured and predicted discharge. The model captures well both peak and magnitude. The largest peak at day 2018-06-01 is overestimated (measured 16.6 m^3/s vs 21.0 m^3/s). At the end of the series, discharge is overestimated by a factor 2.2.

Another peculiarity is the intermittent flat line between 2018-06-20 to 2018-08-10. This is the lowest value (0.84 m^3/s) the model can predict. Note that XGBoost essentially places all samples in bins, meaning there is a certain output range XGBoost can predict in. This might be explained by the training process. Because of the squared error loss function, getting one peak correct is prioritized over reducing the squared error of low flows. For example, the squared error at the largest peak is 19.36 $(m^3/s)^2$. Looking at it from loss function perspective, it is much more beneficial to reduce this number compared to a few low flow errors of 0.16 or 0.09 $(m^3/s)^2$.

Another striking observation is the 95% ensemble interval; it is extremely narrow, except at some peaks. This can be attributed to the way it is calculated, see Section 3.3.4. Apparently, the predictions of the nine different models resulting from k-fold validation are not all that different. This can also be seen in the NSE of the folds (right of Figure 4.2). If the models are not that different, neither will the predictions be different. Note that all

nine models have the same hyperparameters.
The large gap between train and validation NSE indicates that the model might be overfitted. The training, validation and test NSE and MAE are 0.84, 0.46, 0.71 and 0.64, 0.92, 0.75 m^3/s, respectively. Normally, one would like the metric of the validation set to be close to the metric of the training set. This is not the case, and it might be that the algorithm learnt to replicate the output data instead of learning the patterns behind it. However, if this was the case, the test set would perform worse too. As According to the classification given in Table 3.4. this model would classify as good.

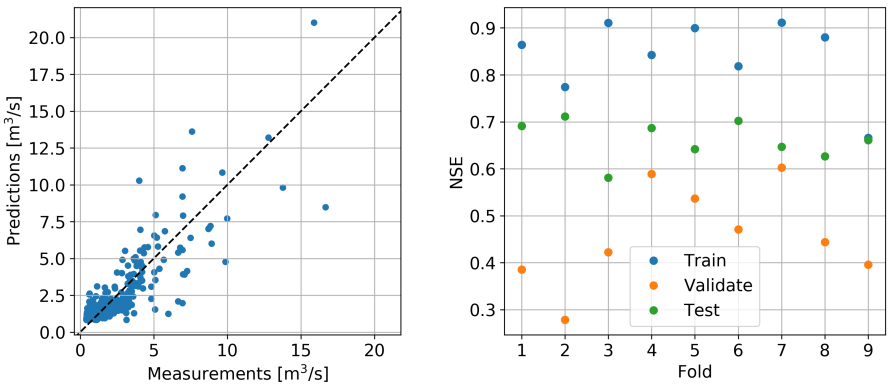

Figure 4.2: Simulated daily maximum discharge using gradient boosting. Left: scatterplot of discharge measurements and predictions. Right: NSE for each fold.

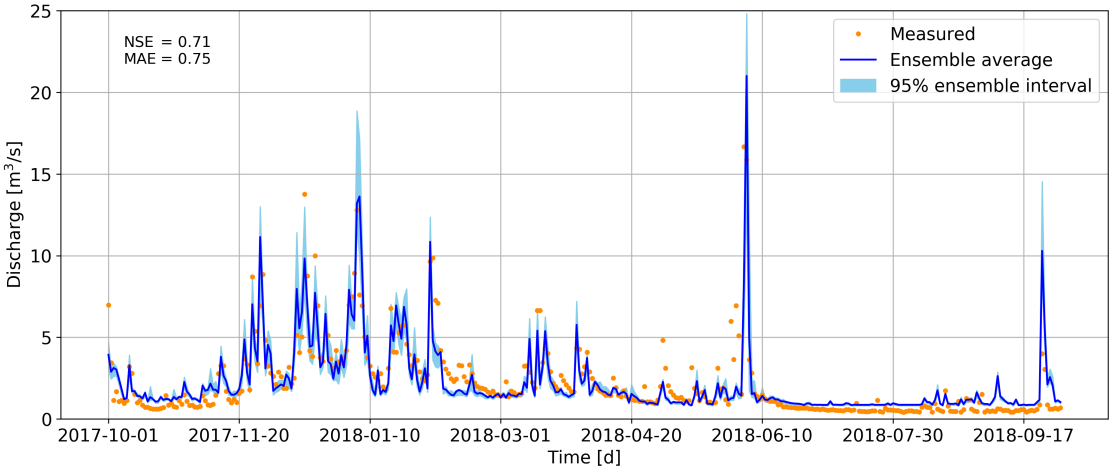

Figure 4.3: Hydrograph of measured discharge and simulated daily maximum discharge using gradient boosting.

DAILY LOGARITHMIC DISCHARGE

A problem with predicting discharge is imbalance of the training set. Low flows occur frequently while higher flows are uncommon. Most ML algorithms tend to predict lower flows, as a unknown flow is more likely to be on the lower side due to this imbalance. A way to counteract this is to decrease the range of flows by modelling the natural logarithm of discharge and then transforming it back to normal discharge after predicting by taking the natural exponent. The distribution of the input discharge, both the 'normal' discharge and discharge transformed using the natural logarithm, are shown in Figure 4.4.
Figure 4.5 and 4.6 show the results of the model trained on the transformed data. The same input variables are used, but with different hyperparameters. MAE is lower as can be expected, simply because the range of the output has decreased. NSE is slightly higher, 0.72 for the logarithm versus 0.71 using normal discharge. Especially low flows are not captured well by the model and generally overestimated. This difference is amplified by the natural logarithm of a value between 0 and 1. The gradient boosting model's lowest possible prediction is 0.82 m^3/s and the lowest measurement is 0.39 m^3/s. Taking the natural logarithm of these two

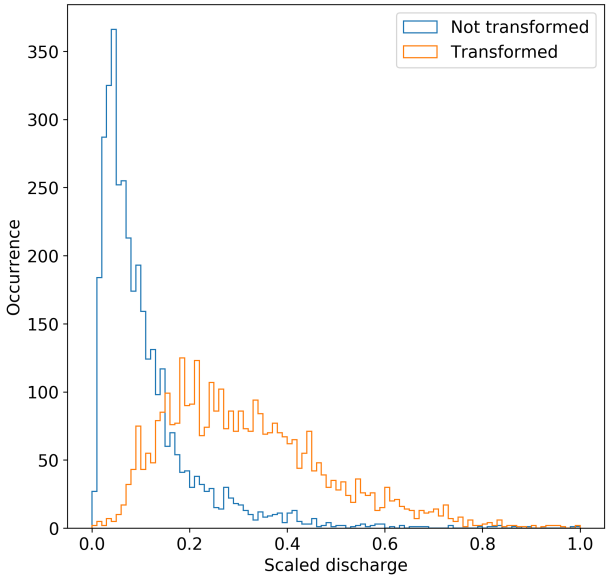

Figure 4.4: Histogram of the 'normal' discharge and discharge transformed using the natural logarithm. Note that the transformed discharge is more evenly distributed.

numbers gives -0.20 and -0.94, respectively. The initial difference was 0.43 m^3/s but has now become 0.74 m^3/s.
What can be seen from the scatterplot in Figure 4.5 is that most points above zero (so discharge above 1 m^3/s) are under the dashed line, meaning most of those points are underestimated.
It becomes clear that predicting the logarithm of the discharge does not improve predictions from Figure 4.7. Here, the values are transformed back by taking the exponent of each prediction and measurement. NSE decreases to 0.66 and MAE increases to 0.77 m^3/s, compared to 0.71 and 0.75 for NSE and MAE without any transformation at all, respectively.

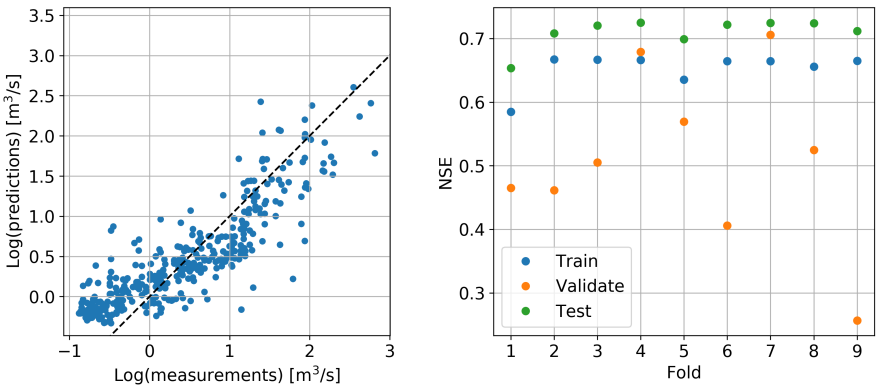

Figure 4.5: Simulated maximum discharge using gradient boosting. Note that it is the logarithm of the discharge. Left: scatterplot of discharge measurements and predictions. Right: NSE for each fold.

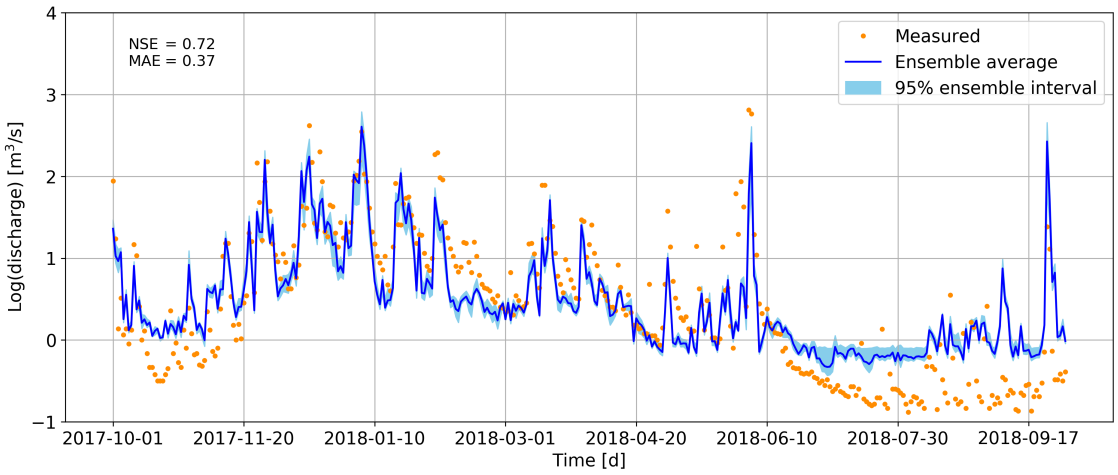

Figure 4.6: Hydrograph of measured discharge and predicted daily maximum discharge of today using gradient boosting. Note that it is the logarithm of the discharge.

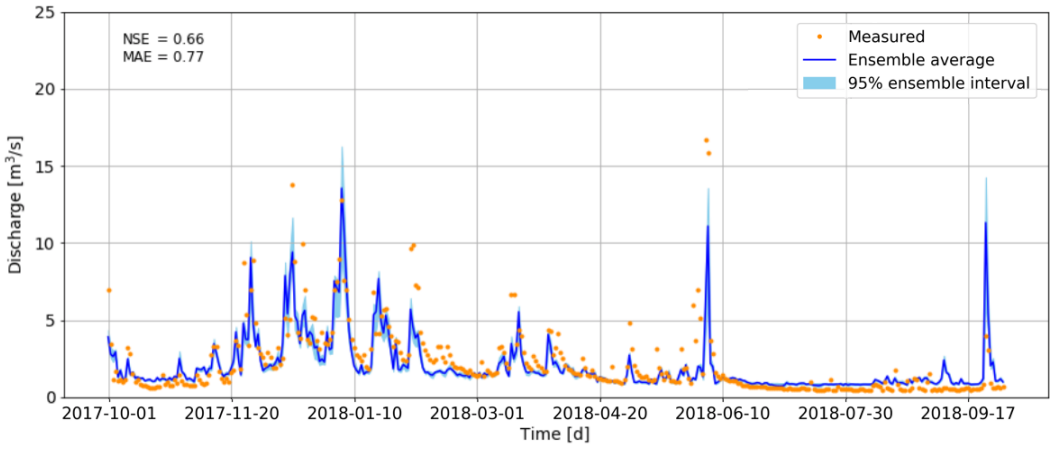

Figure 4.7: Hydrograph of measured discharge and predicted daily discharge of today using gradient boosting. Discharge is transformed back by taking the natural exponent of the logarithmic discharge.

DAILY WATER DEPTH

Water depth is also modelled instead of discharge for the same reason as given above; a smaller prediction range. The same variables are used, but with different hyperparameters. Solely by looking at the scatter plot in Figure 4.8 one can notice that most points are under the dashed line, indicating underestimation of the water depth. This is confirmed by looking at the hydrograph at Figure 4.9. Especially the low flows are underestimated, as well as the falling limbs. The model's performance is sufficient with an NSE of 0.55 and the mean absolute error is 0.07 centimetres. The peak timing is accurate and peak magnitude is also acceptable. Actually, the only difference with predicting discharge in Figure 4.3 is that the output variable is generally underestimated here, where it is overestimated at discharge. The likely explanation is simply the rating curve. If one would apply the rating curve to the predicted water depths, low flows are likely to be overestimated once again. The 95% ensemble interval is narrow, due to the same reason as given before; the k fold models do not differ that much.

Daily water depth has also been modelled using deep learning for both catchments. The results are similar to the result presented here in the sense that it does not outperform modelling discharge. Therefore, these results can be found in Appendix B.5.

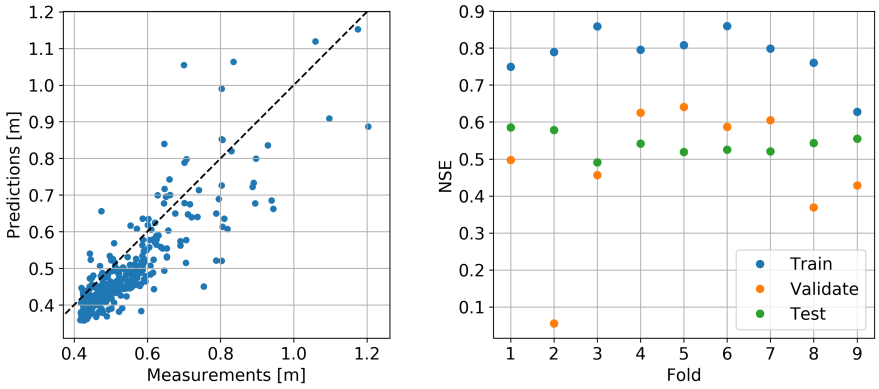

Figure 4.8: Simulated maximum water depth using gradient boosting. Left: scatterplot of water depth measurements and predictions. Right: NSE for each fold.

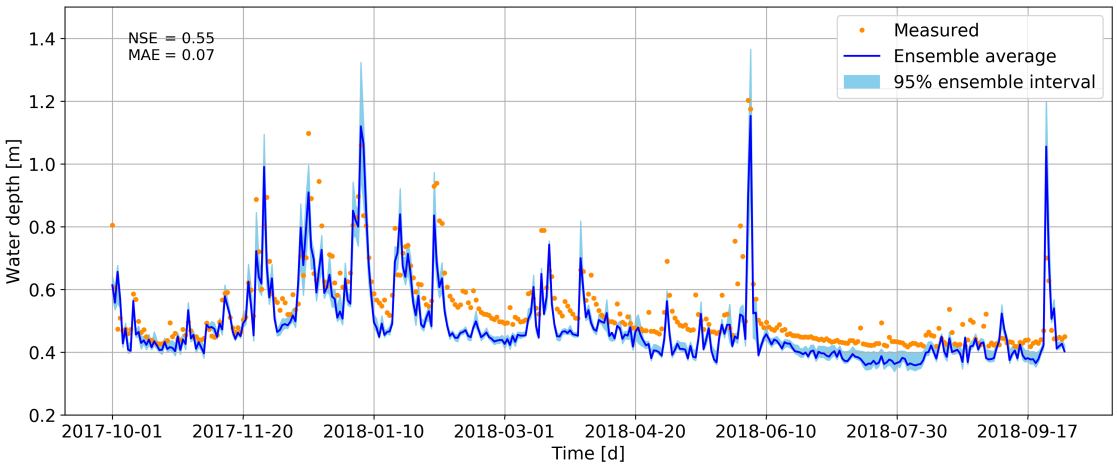

Figure 4.9: Hydrograph of measured water depth and simulated maximum water depth of today using gradient boosting.

HOURLY DISCHARGE

Figure 4.10 and 4.11 show the results of modelling the maximum discharge on a hourly resolution. The same variables are used as before, but with different hyperparameters (see bottom of Table 4.4). Looking at the hydrograph, timing of the peaks is correct, but magnitude is far off (except the largest peak) compared to the daily resolution. This is confirmed by the the scatterplot. The close to horizontal lines are measured peaks that are simulated at a nearly constant discharge (e.g. at predicted discharge of 6.5, 7.6 and 16 m^3/s). The curve above 10 m^3/s on the y-axis, and the curve starting at 5 m^3/s on the x-axis display the rising and falling limb of the largest peak (occurring at approximately 2018-06-09 hours), respectively. It shows that most higher flows are either over- or underestimated. The test set NSE of 0.58 indicates sufficient model performance. NSE of the hourly model is lower than the daily model (0.71), but so is MAE (0.60 vs 0.75 m^3/s). Confidence intervals are narrower on a hourly resolution as well. It can be concluded that the daily model performs better.

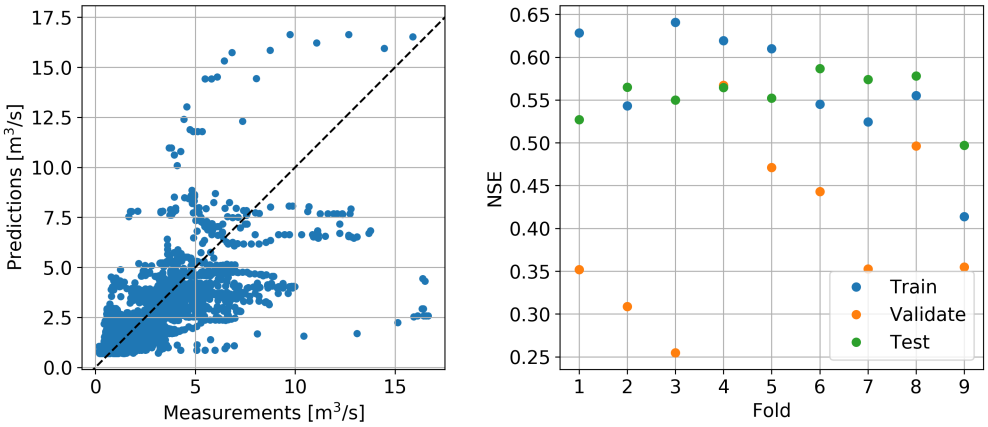

Figure 4.10: Simulated hourly maximum discharge using gradient boosting. Left: scatterplot of hourly discharge measurements and predictions. Right: NSE for each fold.

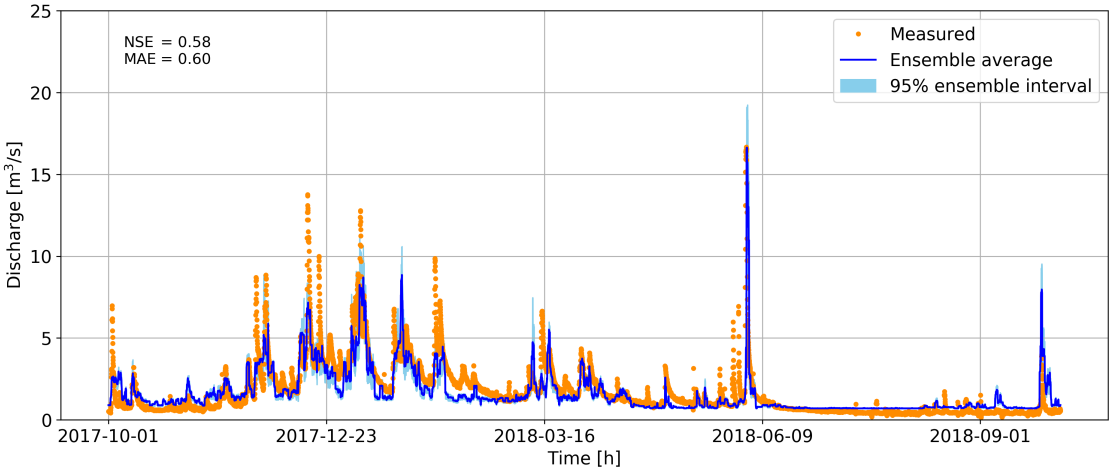

Figure 4.11: Hydrograph of measured hourly discharge and simulated hourly maximum discharge using gradient boosting.

FORECASTING DAILY DISCHARGE

Where the previous models are concerned with modelling today's maximum discharge, the results in the remaining part of this subsection are concerned with forecasting maximum daily discharge. The same variables are used, but hyperparameters differ. Figure 4.12 and 4.13 show the scatterplot and hydrograph of the one-day-ahead forecast done for each consecutive day in a year. The NSE decreases from 0.71 to 0.57 and MAE increases from to 0.75 to 0.81 m^3/s compared to the daily simulations. This model would classify as sufficient, if solely based on NSE. Timing of the peaks is still accurate. Magnitude is underestimated but for two exceptions; the forecast peak just before 2018-06-10 and the modelled peak just after 2018-09-17. Both are overestimated, the latter gravely. This is also the case when predicting today's discharge (Figure 4.3. In addition, when comparing these two hydrographs it is clear that, except those two points, the forecast hydrograph is slightly dampened. As the forecast horizon increases this dampening becomes stronger, ultimately converging to the mean of the training set. This seems reasonable. For example, if one would forecast the discharge over 100 days with today's data, the algorithm is unlikely to find a relationship. Therefore, its best guess is the mean of the training data.

When forecasting two or three days ahead, performance drops to a NSE of 0.41 and 0.37, respectively. MAE increases to 0.99 and 1.05 m^3/s, the hydrographs and scatterplots can be found in Appendix B. Based on these results, it can be concluded that model performance of forecasting more than one day ahead on a daily time scale is insufficient for the Geul when using gradient boosting.

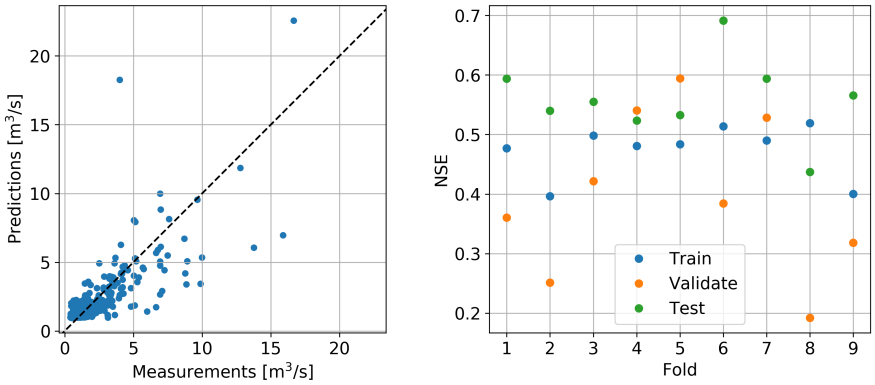

Figure 4.12: 1-day-ahead forecast of daily maximum discharge using gradient boosting. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

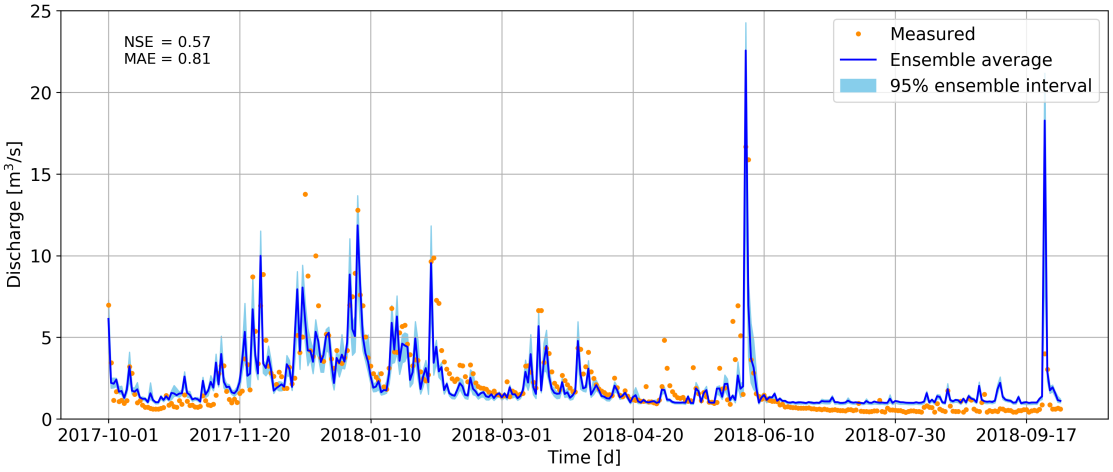

Figure 4.13: Hydrograph of measured discharge and 1-day-ahead forecast of daily maximum discharge using gradient boosting.

FORECASTING HOURLY DISCHARGE

Figure 4.14 shows scatterplots of the 6, 12, 24 and 48-hour-ahead forecast. It shows that as forecasting time increases, the forecast values dampen. This is also confirmed by looking at the last column of Table 4.3, showing the mean forecast. The mean decreases towards the mean of the training set which is 1.55 m^3/s. Peculiar is that the 24-hours-ahead forecast has a higher NSE (0.54) than the 12-hours-ahead discharge prediction (0.50). This might be caused by the hyperparameter tuning. All models are run 50 times with randomly selected hyperparameters and the best performing is selected. While this generally results in being close to the best possible performance, it does not necessarily result in the best performance. Therefore, the 24-hour-ahead forecast might have ended up with closer to the perfect hyperparameters, compared to the 12-hour-ahead forecast. Running both models more than 50 times is likely to solve this peculiarity.
To summarize, the forecast up to 24 hours ahead performs sufficiently. If the forecast horizon increases above 24 hours, the timing of the peaks is still okay but the magnitude is severely underestimated.

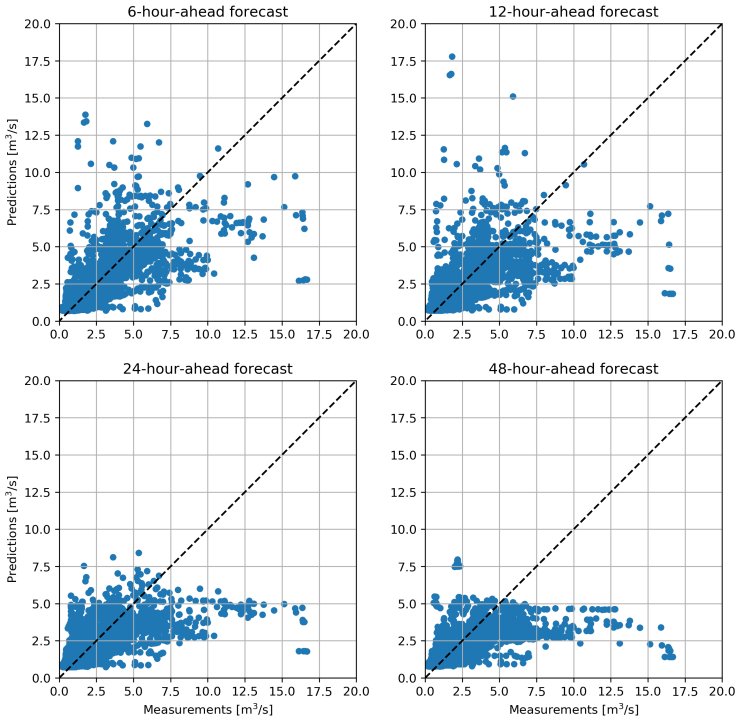

Figure 4.14: Scatterplot of the 6, 12, 24 and 48-hour-ahead forecasts.

Table 4.3: NSE, MAE (in m^3/s) and mean (in m^3/s) of different forecast horizons.

Forecast horizon	Test NSE	Test MAE	Test mean
6-hours	0.57	0.60	1.68
12-hours	0.50	0.62	1.61
24-hours	0.54	0.64	1.63
48-hours	0.48	0.65	1.59

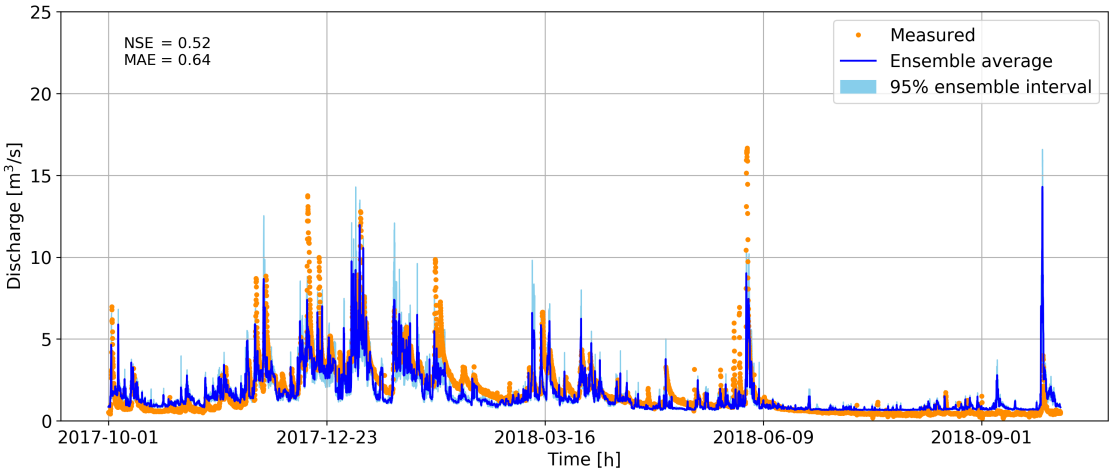

Figure 4.15: Hydrograph of measured discharge and 24-hour-ahead forecast of hourly maximum discharge using gradient boosting.

Table 4.4: Overview of performance and hyperparameters for all gradient boosting models for the Geul catchment. Maximum daily discharge is modelled unless stated otherwise.

	NSE			MAE			Hyperparameters				
Model	train	val	test	train	val	test	Colsample	η	max_depth	n_est	subsample
Simulation	0.84	0.46	0.71	0.64	0.92	0.75	0.898	0.0139	6	1188	0.923
Logarithm of discharge	0.65	0.51	0.72	0.34	0.38	0.37	0.768	0.0237	4	1151	0.514
Daily water depth	0.78	0.47	0.55	0.05	0.07	0.06	0.868	0.0183	6	856	0.600
1-day-ahead	0.56	0.38	0.57	0.82	0.93	0.81	0.637	0.0137	5	717	0.769
2-days-ahead	0.22	0.23	0.41	1.07	1.06	0.99	0.996	0.0250	5	1031	0.552
3-days-ahead	0.14	0.18	0.37	1.23	1.17	1.05	0.961	0.0127	4	1045	0.699
Hourly	0.56	0.40	0.58	0.58	0.61	0.60	0.973	0.0103	4	931	0.975
6-hours-ahead	0.48	0.36	0.57	0.61	0.65	0.60	0.768	0.0234	4	1114	0.514
12-hours-ahead	0.37	0.33	0.50	0.64	0.65	0.65	0.961	0.0127	4	1065	0.696
24-hours-ahead	0.34	0.32	0.54	0.65	0.66	0.64	0.758	0.0293	4	588	0.568
48-hours-ahead	0.26	0.29	0.48	0.67	0.65	0.65	0.961	0.0127	4	1065	0.699

Table 4.5: Overview of performance and hyperparameters for all gradient boosting models for the Rur catchment. Maximum daily discharge is modelled unless stated otherwise.

	NSE			MAE			Hyperparameters				
Model	train	val	test	train	val	test	Colsample	η	max_depth	n_est	subsample
Simulation	0.54	0.33	0.51	5.50	6.35	5.63	0.822	0.0077	4	521	0.522
Logarithm of discharge	0.57	0.37	0.47	0.24	0.27	0.24	0.670	0.0116	4	1209	0.672
Daily water depth	0.55	0.36	0.47	0.15	0.17	0.15	0.915	0.0058	4	1033	0.542
1-day-ahead	0.45	0.30	0.42	5.77	6.45	6.02	0.742	0.0057	4	1146	0.515
2-days-ahead	0.48	0.28	0.35	5.76	6.68	6.30	0.629	0.0292	5	603	0.626
3-days-ahead	0.44	0.26	0.33	5.94	6.75	6.42	0.646	0.0287	5	978	0.704

4.2.2. RUR

In this section the results of the gradient boosting models for the Rur will be discussed. Only daily resolution is modelled as only daily data is used for this catchment. Daily simulations for discharge, the natural logarithm of the discharge and water depth are discussed here, as well as daily discharge forecasts.

DAILY DISCHARGE

By observing the scatterplot of Figure 4.16 one can see that discharge is generally underestimated, as most points are below the dashed line. This becomes clear by looking at the hydrograph in Figure 4.17. While peak timing is good, magnitude is underestimated. Especially the period between 2016-06-10 and 2016-07-20 is gravely underestimated. This might be due to the reservoirs slowly releasing peak flow from upstream but further analysis is needed to verify this. The simulation follows the pattern of the measurements but magnitude is inaccurate. However, with an NSE of 0.51 the model's performance is still sufficient.

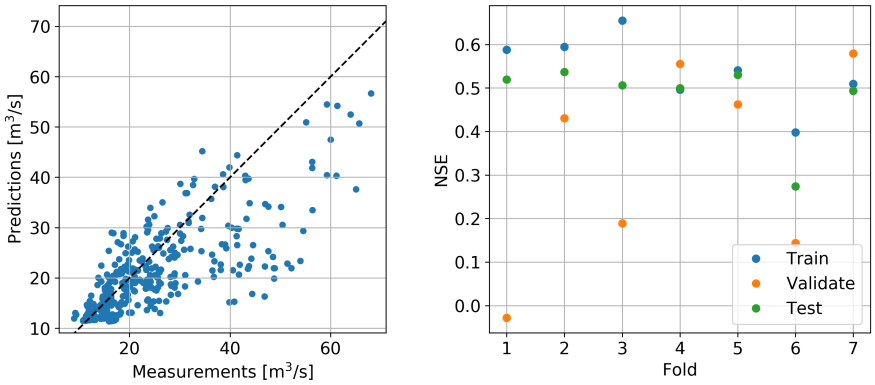

Figure 4.16: Simulated daily maximum discharge using gradient boosting. Left: scatterplot of discharge measurements and predictions. Right: NSE for each fold.

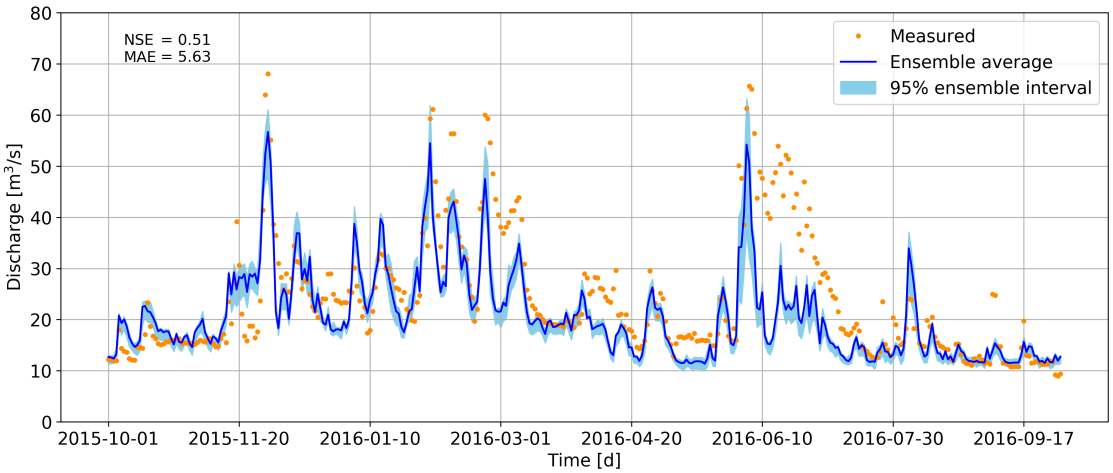

Figure 4.17: Hydrograph of measured discharge and simulated daily maximum discharge using gradient boosting.

DAILY LOGARITHMIC DISCHARGE

For the same reasons as given in section 4.2.1, the natural logarithm of the discharge is modelled and then transformed back by taking the natural exponent. Looking at the metrics, NSE decreases from 0.51 to 0.43 compared to modelling discharge without transformations. Mean absolute error increases from 5.63 to 6.02 m^3/s. Peak timing is still accurate, but magnitude of the flow is worse. The gradient boosting model for the Geul showed similar results, indicating that transforming discharge using the natural logarithm does not improve performance, but rather decreases it.

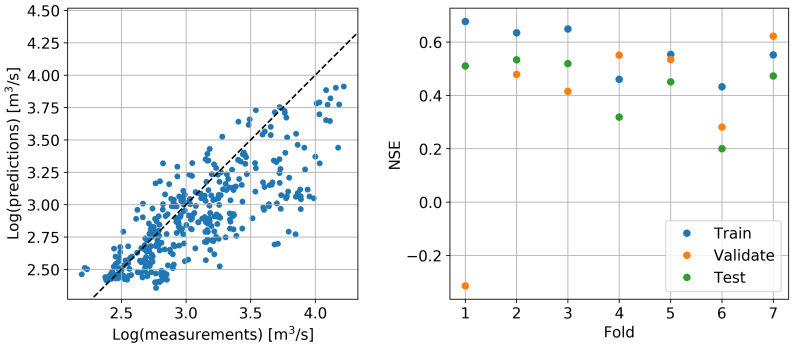

Figure 4.18: Simulated maximum discharge using gradient boosting. Note that it is the logarithm of the discharge. Left: scatterplot of discharge measurements and predictions. Right: NSE for each fold.

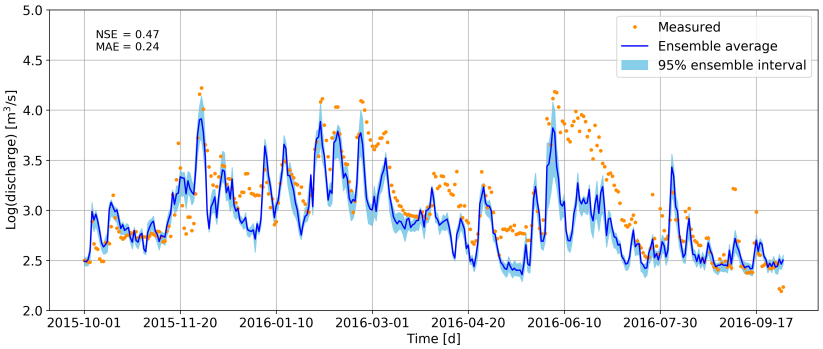

Figure 4.19: Hydrograph of measured discharge and predicted daily maximum discharge of today using gradient boosting. Note that it is the logarithm of the discharge.

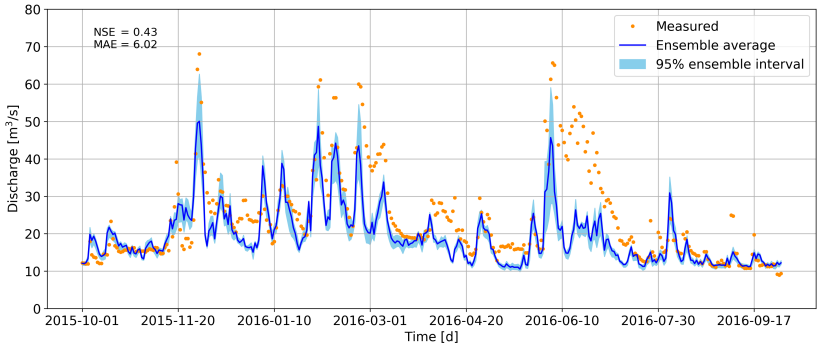

Figure 4.20: Hydrograph of measured discharge and predicted daily discharge of today using gradient boosting. Discharge is transformed back by taking the natural exponent of the logarithmic discharge.

FORECASTING DAILY DISCHARGE

When forecasting instead of simulating performance decreases as expected. NSE drops to 0.42 from 0.51 compared to the daily maximum discharge simulation, and MAE increases from 5.63 to 6.03 m^3/s. However, peak timing is still accurate and magnitude is still quite close to the measurements. Looking purely at NSE, this model's performance is classified as insufficient as it is below 0.50.

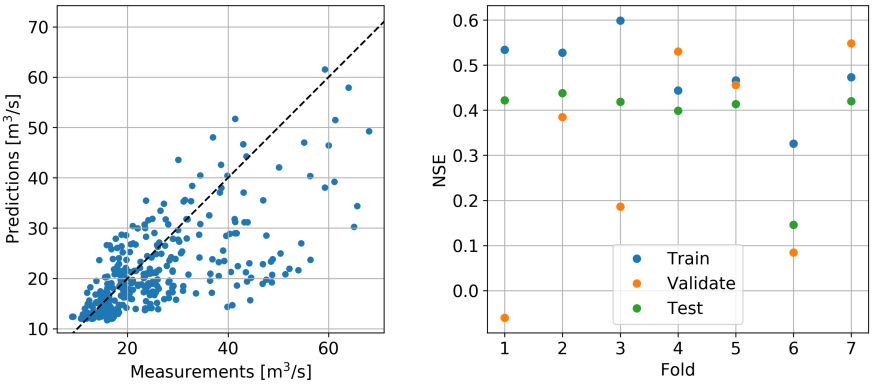

Figure 4.21: 1-day-ahead forecast of daily maximum discharge using gradient boosting. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

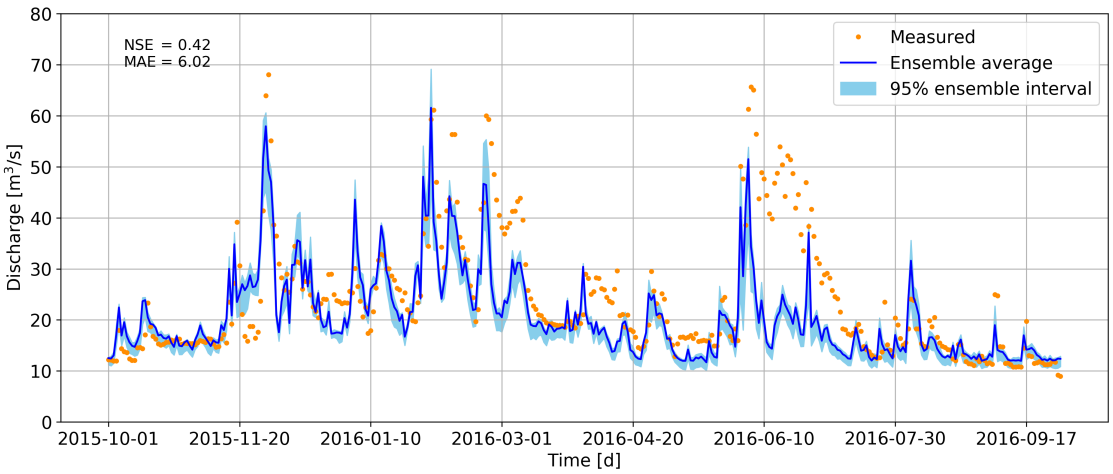

Figure 4.22: Hydrograph of measured discharge and 1-day-ahead forecast of daily maximum discharge using gradient boosting.

4.2.3. LIMITATIONS

Like most ML models, XGBoost cannot extrapolate. The model's output is constrained to the training range of the output variable. Hyperparameter optimization is now done using random search, which gives good results but not necessarily the best results. Performing an exhaustive grid search or Bayesian optimization is likely to improve performance.

Modelling discharge on a hourly time scale introduces some variable issues. If one wants to have the evaporation of the past two days, as in the daily model, one needs to add 48 new variables. Now the evaporation of the past two hours is taken into account, removing those two additional variables likely will not decrease performance.

The ensemble interval is narrow. As was mentioned before, this is because the nine different k-fold models do not differ as much. A better way might be to use bootstrap aggregating (bagging) to create the nine different training and validation sets. This method introduces randomness to the sets by randomly sampling with replacement. This is likely to result in more diverse training and validation sets.

4.3. DEEP LEARNING

The results of the deep learning models are presented and discussed here. Results for discharge on a daily and hourly resolution are shown, as well as logarithmic discharge. Again, discharge based on that same day's variables is discussed before the forecasting part.
Tuning the hyperparameters manually resulted in a input step size of four steps, a batch size of 64, and two hidden layers consisting of 64 and 8 LSTM neurons, respectively. Both hidden layers use the ReLu activation function. The output layer uses the sigmoid activation function. This network is used for all deep learning models, for both Geul and Rur.

4.3.1. GEUL

Before each model is discussed in depth, please note that an outlier is removed, as mentioned in Section 4.2.

DAILY DISCHARGE

The left of Figure 4.23 shows a scatterplot of the predicted and measured discharge. What can be seen in the scatterplot is that higher flows (> 10 m^3/s) are slightly underestimated except the largest peak. This is not surprising as it is difficult to accurately model discharge peaks, as discussed in Section 4.2.
Figure 4.24 shows the hydrograph of the measured and predicted daily maximum discharge. The timing of the peaks is right. Low flows are captured well by the model, whereas gradient boosting models show a horizontal line.
The 95% ensemble interval seems pretty reliable. It is larger for higher discharges, which is what one would expect. With an NSE of 0.69 the model's performance is qualified as good.

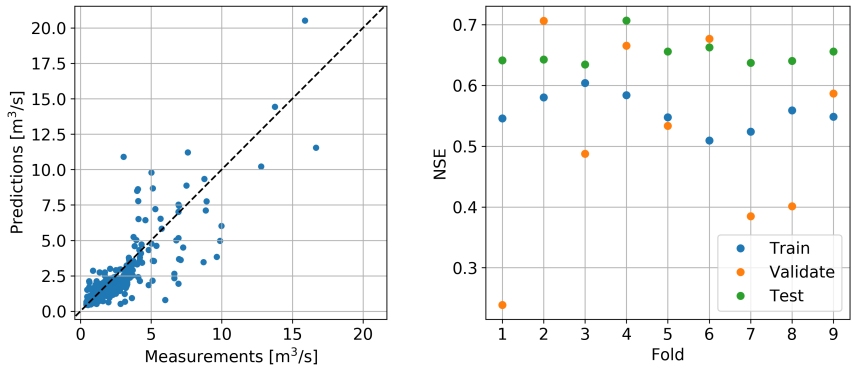

Figure 4.23: Simulated daily maximum discharge of today using deep learning. Left: scatterplot of discharge measurements and predictions. Right: NSE for each fold.

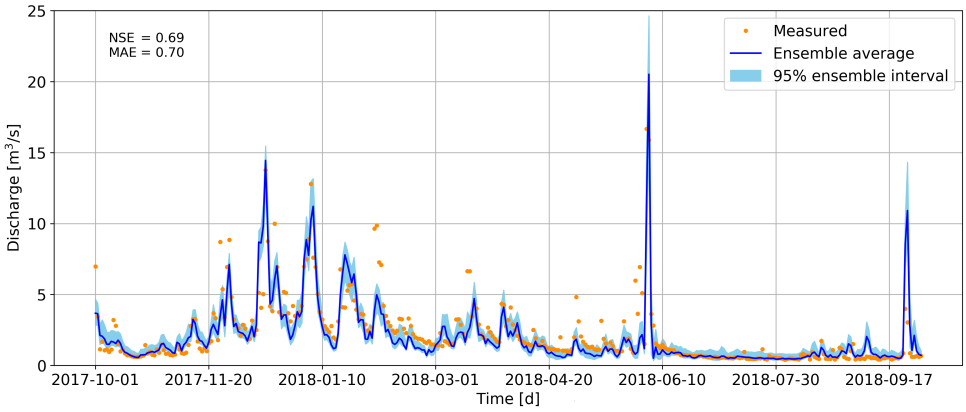

Figure 4.24: Hydrograph of measured discharge and simulated daily maximum discharge using deep learning.

DAILY LOGARITHMIC DISCHARGE

As mentioned in Section 4.2.1, taking the logarithm of a variable to reduce its range, and then transforming it back might increase performance. Figure 4.25 and 4.26 show the results of the discharge transformed before training. MAE is lower as can be expected, simply because the range of the output has decreased. NSE is slightly higher, 0.74 for the logarithm versus 0.69 using normal discharge. Especially low flows are not captured well by the model and generally overestimated. The reasoning given in Section 4.2.1 applies here too. Contrary to gradient boosting, performance slightly increases, as the values are transformed back by taking the natural exponent, as can be seen from Figure 4.27. NSE increases to 0.71 from 0.69 when discharge is not transformed. MAE remains the same at 0.70 m^3/s.

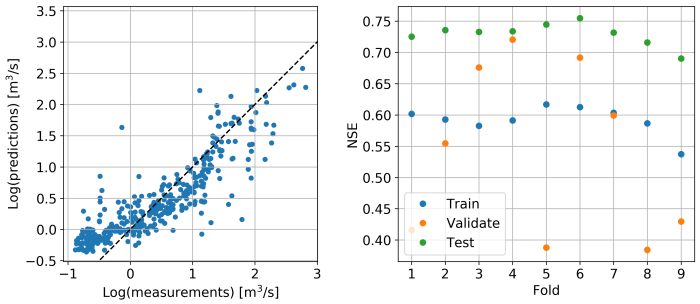

Figure 4.25: Simulated maximum discharge of today using deep learning. Note that it is the logarithm of the discharge. Left: scatterplot of water discharge measurements and predictions. Right: NSE for each fold.

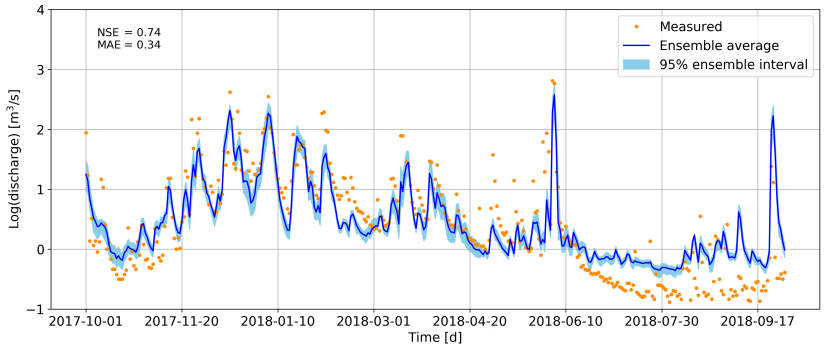

Figure 4.26: Hydrograph of measured discharge and simulated daily maximum discharge of today using deep learning. Note that it is the logarithm of the discharge.

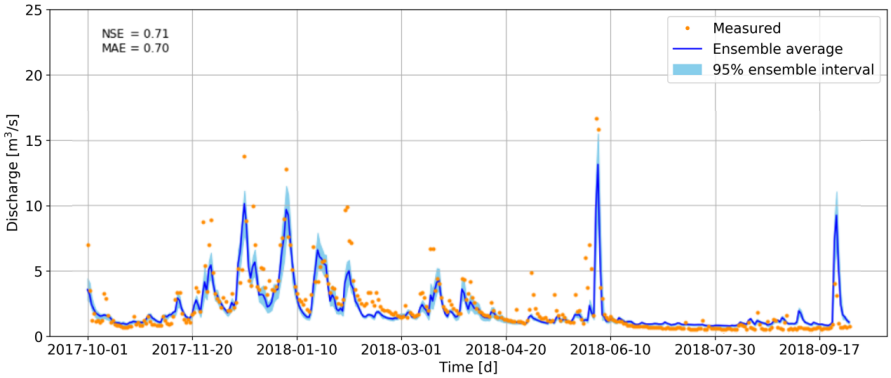

Figure 4.27: Hydrograph of measured discharge and simulated daily maximum discharge of today using deep learning. Discharge is transformed back by taking the natural exponent of the logarithmic discharge.

HOURLY DISCHARGE

The scatterplot of measured and predicted hourly maximum discharge is given in Figure 4.28. The first peculiarity to notice are the horizontal lines at higher measured discharges (e.g. at 2.6 and 4.8 m^3/s). These are, as was the case with the hourly XGBoost model, falling and rising limbs of the higher discharges. These are more difficult to model, probably due to class imbalance in the training set meaning there are simply much more low flows than high flows. Modelling the lower flows more accurately is better than to estimate higher discharge correctly, even when training with mean squared error. On the right of the same figure, the NSE is given for each fold. All test set values are higher than the training set NSEs. This can happen if the model generalizes well and if the training set is large compared to the test set (ratio of 9:1 here) [125].
Hourly predictions are better compared to the daily discharge predictions in terms of MAE (0.54 vs 0.73 m^3/s). NSE is about similar (0.66 vs 0.69). Peak timing is good but magnitude is often underestimated. The confidence interval is narrower too, compared to the daily prediction. Low flows are simulated well.

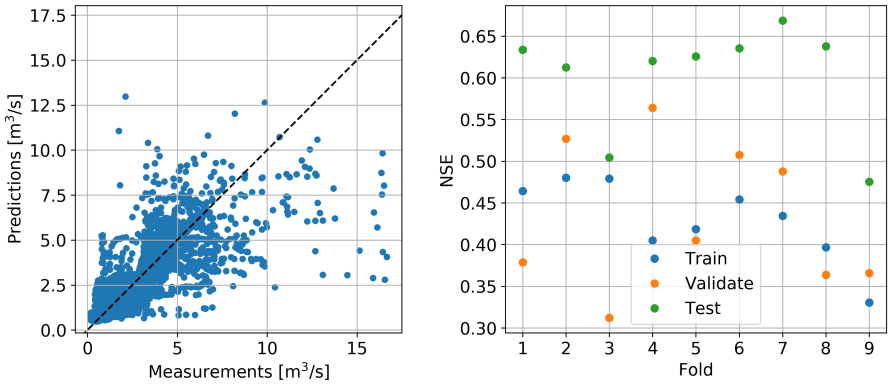

Figure 4.28: Simulated hourly maximum discharge using deep learning. Left: scatterplot of discharge measurements and predictions. Right: NSE for each fold.

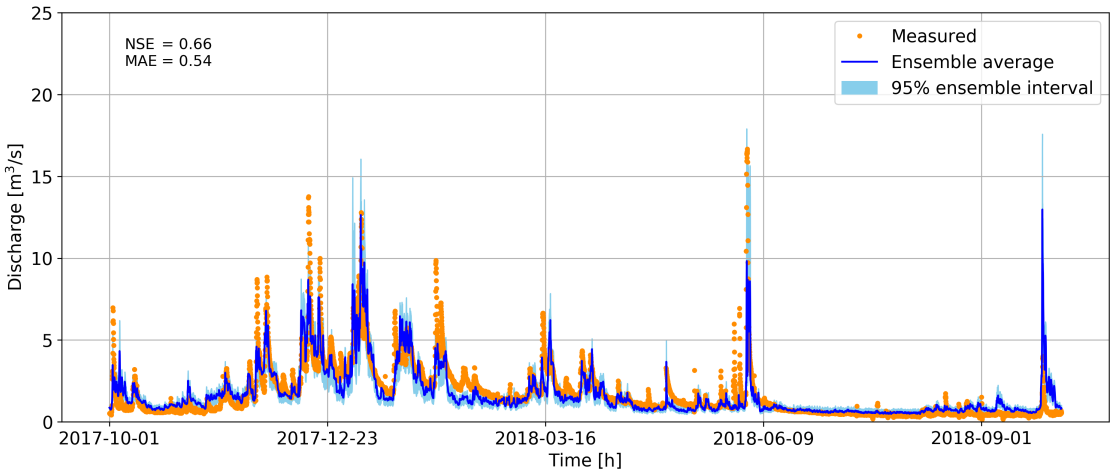

Figure 4.29: Hydrograph of measured discharge and simulated hourly maximum discharge using deep learning.

FORECASTING DAILY DISCHARGE

Where the previous models are concerned with modelling today's discharge, the results here are about forecasting maximum daily discharge. The variables and network configuration remain the same as for the daily simulations. Figure 4.30 and 4.31 show the scatterplot and hydrograph of the 1-day-ahead forecast done for each consecutive day in a year. The NSE decreases from 0.69 to 0.58 and MAE increases from to 0.70 to 0.78 m^3/s. This model would classify as sufficient, if solely based on NSE. Timing of the peaks is accurate. Magnitude is often slightly underestimated except for the largest peak and the high at the end of the series. Low flows are forecast well. Compared with the daily simulation, the ensemble intervals are slightly wider which is what one would expect as information about tomorrow is missing.
When forecasting two or three days ahead, NSE decreases to 0.41 and 0.37, respectively. MAE increases to 0.99 and 1.05 m^3/s.

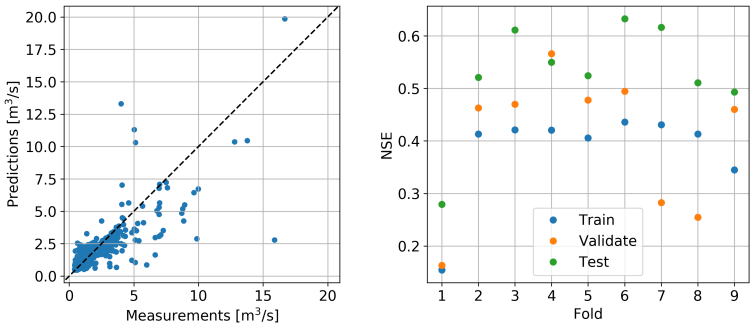

Figure 4.30: 1-day-ahead forecast of daily maximum discharge using deep learning. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

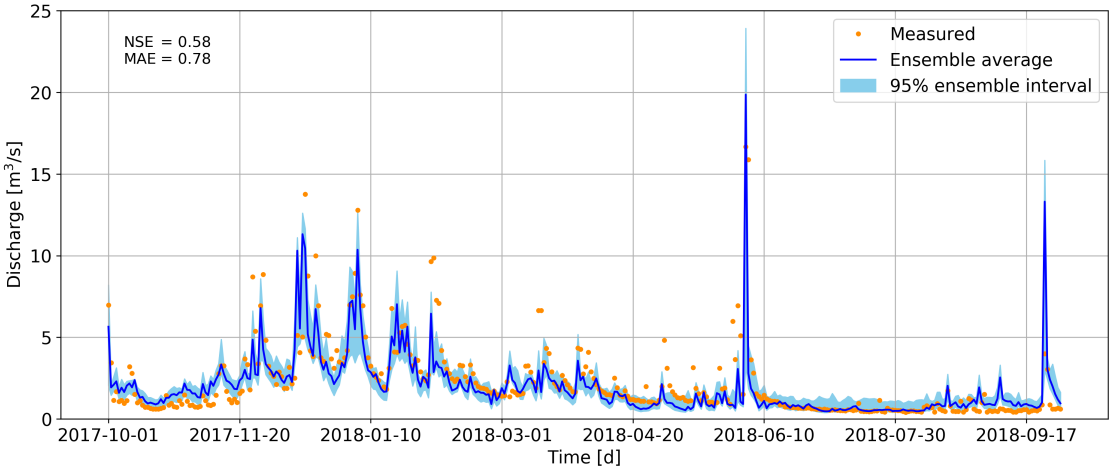

Figure 4.31: Hydrograph of measured discharge and 1-day-ahead forecast of daily maximum discharge using deep learning.

FORECASTING HOURLY DISCHARGE

The results of forecasting maximum hourly discharge 12-hours-ahead are given in Figure 4.32 and 4.33. Starting with the metrics, NSE decreases from 0.66 to 0.55 compared to the hourly simulation. MAE increases from 0.54 to 0.61 m^3/s. Peak timing is still on point. In both the simulation as the forecast, the last peak is gravely overestimated. The period between 2017-12-15 and 2018-01-10 is more noisy for the forecast. Again, this is not strange as information is missing compared to simulating.

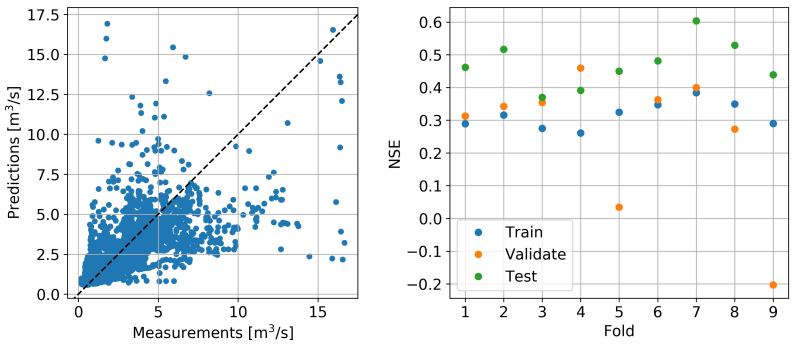

Figure 4.32: 12-hours-ahead forecast of hourly maximum discharge using deep learning. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

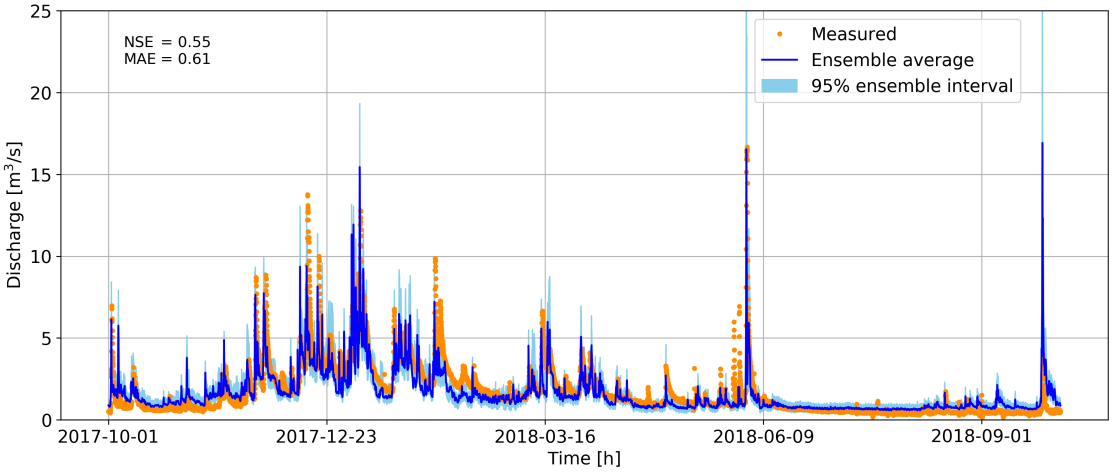

Figure 4.33: Hydrograph of measured discharge and 12-hours-ahead forecast of hourly maximum discharge using deep learning.

Table 4.6: Overview of performance for all deep learning models for the Geul catchment. Daily maximum discharge is modelled unless stated otherwise.

	NSE			MAE		
Model	train	val	test	train	val	test
Discharge	0.56	0.52	0.69	0.93	0.86	0.70
Logarithm of discharge	0.59	0.54	0.74	0.36	0.35	0.34
Daily water depth	0.52	0.48	0.53	0.07	0.06	0.07
1-day-ahead forecast	0.48	0.47	0.58	0.98	0.87	0.78
2-days-ahead forecast	0.32	0.34	0.46	1.10	1.02	0.88
3-days-ahead forecast	0.13	0.13	0.29	1.24	1.16	0.98
Hourly discharge	0.43	0.43	0.66	0.64	0.57	0.54
12-hours-ahead forecast	0.32	0.26	0.55	0.71	0.62	0.61

Table 4.7: Overview of performance for all deep learning models for the Rur catchment. Daily maximum discharge is modelled unless stated otherwise.

	NSE			MAE		
Model	train	val	test	train	val	test
Discharge	0.41	0.38	0.51	6.85	6.63	5.49
Logarithm of discharge	0.49	0.41	0.56	0.27	0.27	0.22
Daily water depth	0.39	0.35	0.45	0.18	0.17	0.15
1-day-ahead forecast	0.40	0.41	0.45	7.01	6.91	5.83
2-days-ahead forecast	0.31	0.29	0.37	6.30	6.34	6.19
3-days-ahead forecast	0.24	0.30	0.30	7.08	6.78	6.36

4.3.2. RUR

In this section the results of the deep learning models for the Rur will be discussed. Only daily resolution is modelled as only daily data is used for this catchment. Daily simulations for discharge, the natural logarithm of the discharge and water depth are discussed here, as well as daily discharge forecasts.

DAILY DISCHARGE

From the scatterplot in Figure 4.34 it can be seen that the daily maximum discharge is generally underestimated. Peak timing is accurate but magnitude is underestimated. The period 2016-06-10 to 2016-07-20 is again gravely underestimated. The results are actually quite similar to the gradient boosting results of the Rur. Both models have the same NSE (0.51) but MAE is lower for the deep learning model (5.49 vs 5.63 m^3/s). The ensemble interval is larger but this is to be expected as the gradient boosted model has only 7 runs whereas the deep learning model has 35. What is peculiar is that the ensemble interval is equally large at peaks as at lower flows. Intuitively, the interval should be larger for higher discharges as they contain more uncertainty, as discussed in Section 4.2.1.

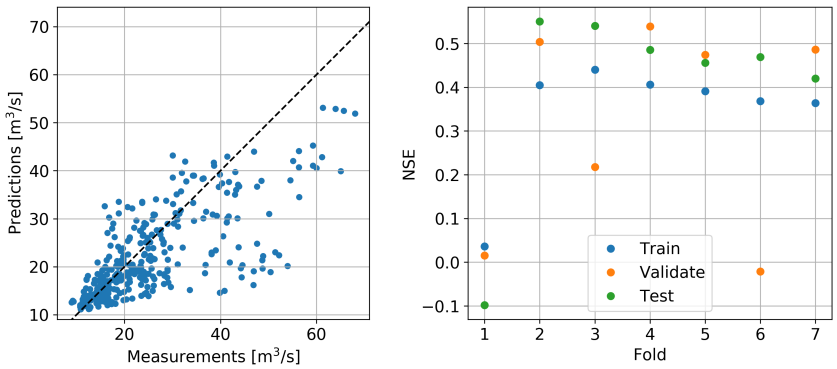

Figure 4.34: Simulated daily maximum discharge of today using deep learning. Left: scatterplot of discharge measurements and predictions. Right: NSE for each fold.

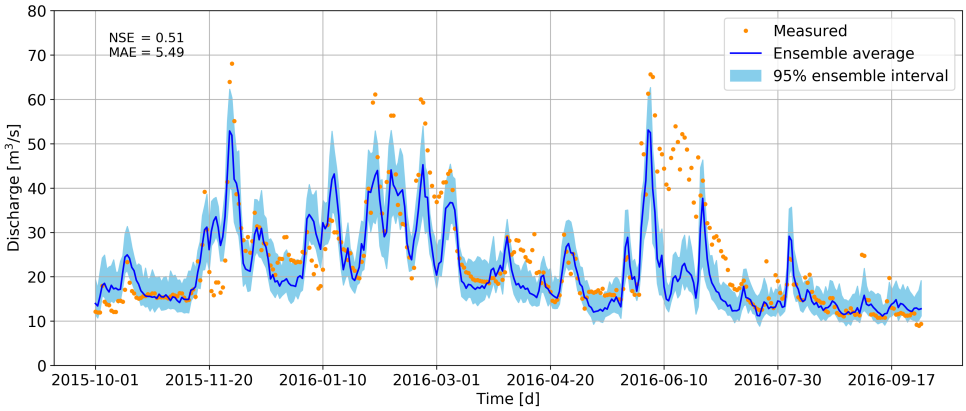

Figure 4.35: Hydrograph of measured discharge and simulated daily maximum discharge using deep learning.

DAILY LOGARITHMIC DISCHARGE

The NSE of both the natural logarithmic and re-transformed discharge is higher than the untransformed discharge, being 0.56, 0.52 and 0.51, respectively. However, MAE slightly increases from 5.49 to 5.53 m^3/s when comparing the untransformed and the re-transformed simulations. With this negligible difference, it can be stated that performance remains the same here, and taking the natural logarithm before training does not affect performance.

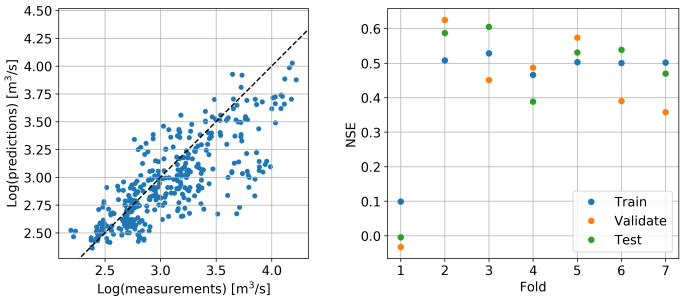

Figure 4.36: Simulated maximum discharge of today using deep learning. Note that it is the logarithm of the discharge. Left: scatterplot of water discharge measurements and predictions. Right: NSE for each fold.

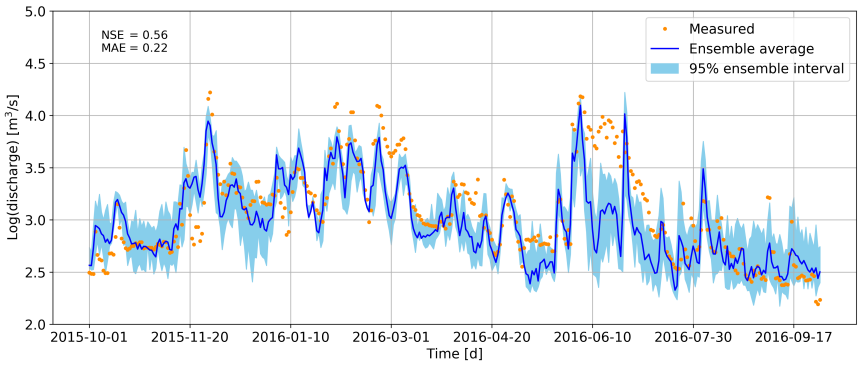

Figure 4.37: Hydrograph of measured discharge and simulated daily maximum discharge of today using deep learning. Note that it is the logarithm of the discharge.

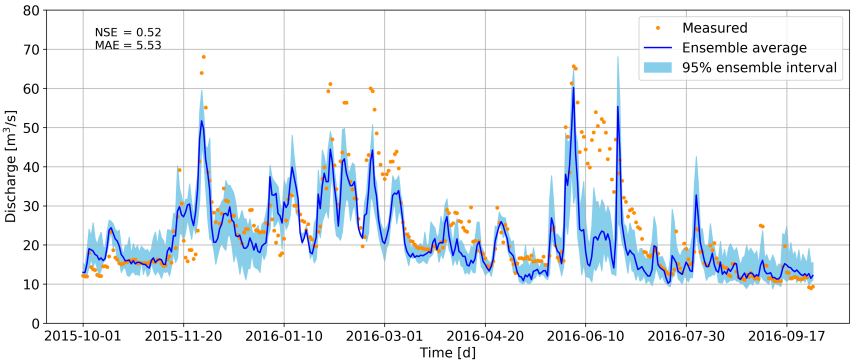

Figure 4.38: Hydrograph of measured discharge and simulated daily maximum discharge of today using deep learning. Discharge is transformed back by taking the natural exponent of the logarithmic discharge.

FORECASTING DAILY DISCHARGE

The hydrograph of the one-day-ahead forecast for daily maximum discharge is given in Figure 4.40. Compared to simulating daily maximum discharge, NSE decreases from 0.51 to 0.45. This is expected as the correlation between in- and output also decreases. The MAE increases to 5.83 from 5.49 m^3/s. Peak timing is still accurate, but slightly more underestimated compared to the simulations. When forecasting two or three days ahead, NSE drops to 0.37 and 0.30, respectively. Mean absolute error increases to 6.19 and 6.13 m^3/s for the two and three days ahead forecasts, respectively.

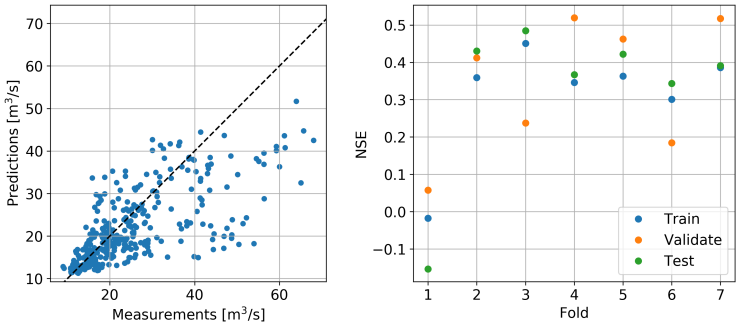

Figure 4.39: 1-day-ahead forecast of daily maximum discharge using deep learning. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

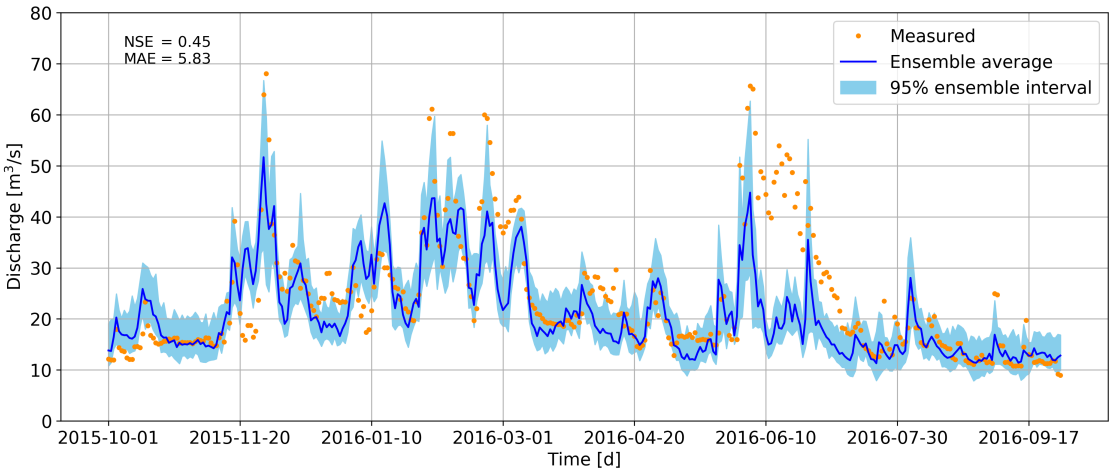

Figure 4.40: Hydrograph of measured discharge and 1-day-ahead forecast of daily maximum discharge using deep learning.

4.3.3. LIMITATIONS

The main limitation of a deep learning model is the amount of hyperparameters to tune. In this study only a couple have been tuned, others have been left at the default value as specified in the programming module. In addition, the network used here has two hidden layers consisting of 64 and 8 LSTM cells respectively. The total network has about 20.000 trainable parameters. Given that on a daily temporal resolution about 3300 samples are available for training, there are roughly six times as much parameters that have to be trained. Because so little data is available (mind, 10 years of data), results can differ quite a lot per training and validation session. Moreover, there is no clear strategy for optimizing hyperparameters, it is either done manually using empirical experience of by a computational demanding grid search or Bayesian optimization.

A second limitation is that neural network cannot extrapolate outside the discharge range in the training set because the values are normalized between the lowest and highest value present. As a result, the maximum output discharge is equal to the maximum training discharge.

Finally, deep learning models require quite some computational effort to train. Another type of recurrent neural network, the Gated Recurrent Unit (GRU), has similar performance but has less gates. This makes the training process faster [126].

4.4. GR4J

In this section the results of the GR4J model are presented. As the model has a daily temporal resolution, results of both ML algorithms are compared to (forecast) daily maximum discharge.

4.4.1. DAILY DISCHARGE

Looking at the hydrograph of the total period, including warm-up and calibration, given in Figure 4.42, one can see that NSE is higher for the test set compared to the calibration period. This is likely due to the lack of high discharges present in the test set. There was one point, as explained earlier on, but has been removed for the GR4J model too to make a fair comparison. Peak timing is good, but magnitude is sometimes underestimated. Looking at the scatterplot (Figure 4.41), there is one point far away from the 1:1 line, which is the largest peak (measured 16.7 m^3/s, predicted 12.1 m^3/s). Low flows are slightly underestimated. The summer of 2018 was exceptionally dry and as no precipitation is fed into the model, the reservoirs will slowly empty. However, in the real world there is still some base flow, probably coming from groundwater. The calibration led to the following parameters; maximum capacity of production store (x_1) is 282 mm; catchment groundwater exchange coefficient (x_2) is 0 mm; one-day-ahead maximum capacity for the routing reservoir (x_3) is 12 mm; and the time base of the unit hydrograph (x_4) is 1.27 days. All parameters fall within the approximate 80% confidence interval given by Perrin et al. [5], except x_3. The confidence interval for that parameter is 20-300 mm. The 12 mm is thus quite low, indicating a small routing reservoir.
With an NSE of 0.65 the model's performance is on the border of sufficient and good.

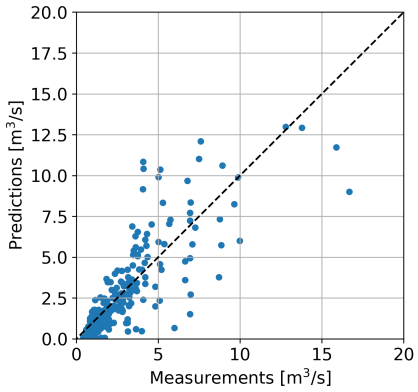

Figure 4.41: Scatterplot of discharge measurements and predicted daily maximum discharge of today.

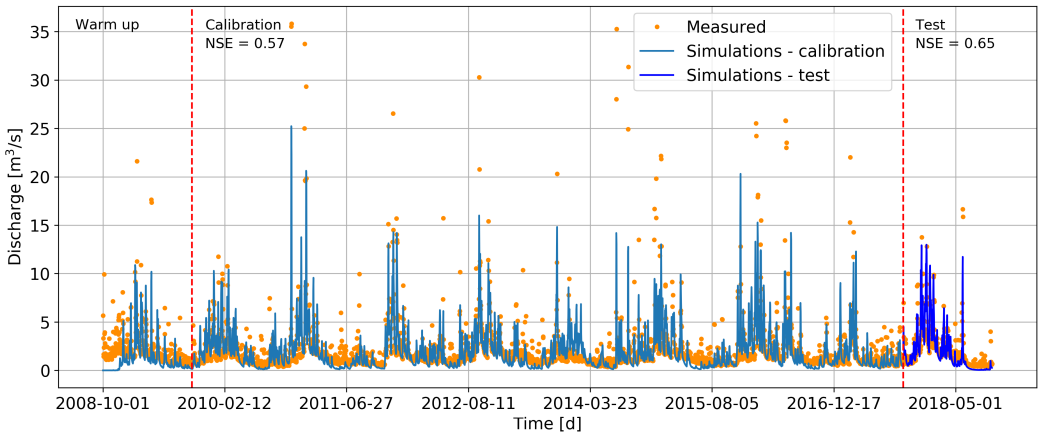

Figure 4.42: Hydrograph of measured discharge and today's daily maximum discharge, including warm-up and calibration periods.

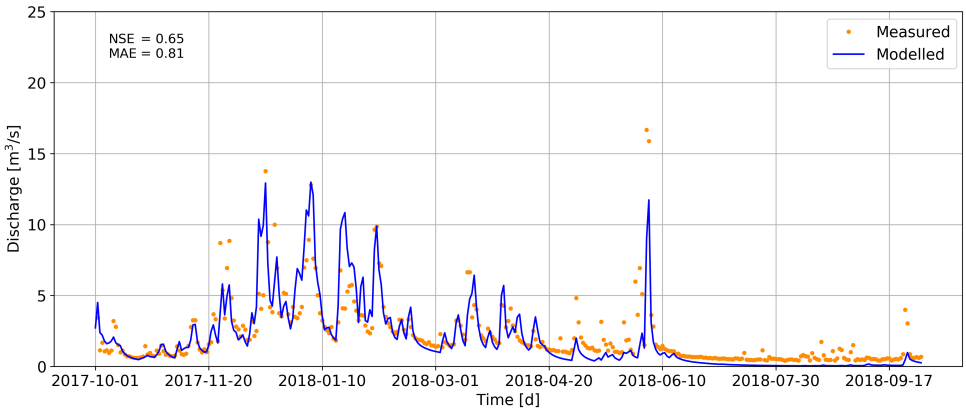

Figure 4.43: Hydrograph of measured discharge and today's daily maximum discharge, test set only.

4.4.2. FORECASTING DAILY DISCHARGE

As expected NSE and MAE worsen when forecasting one day into the future. NSE decreases to 0.51 and MAE increases to 0.84 m^3/s. Magnitude of the largest peak is estimated more accurately when forecasting one day ahead. Low flows are still underestimated, for the same reason as given before. The calibration led to the following parameters; maximum capacity of production store (x_1) is 504 mm; catchment groundwater exchange coefficient (x_2) is 0.1 mm; one-day-ahead maximum capacity for the routing reservoir (x_3) is 7 mm; and the time base of the unit hydrograph (x_4) is 1.1 days. All parameters fall within the approximate 80% confidence interval given by Perrin et al. [5], except x_3. The confidence interval for that parameter is 20-300 mm. The 12 mm is thus quite low, indicating a small routing reservoir.
With an NSE of 0.51 the model's performance is barely sufficient.

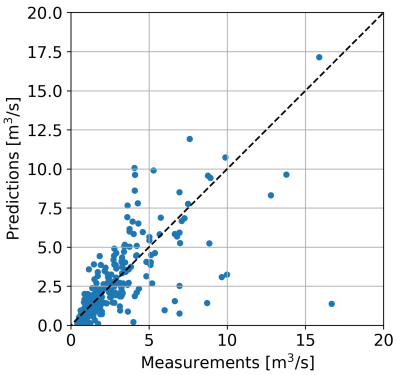

Figure 4.44: Scatterplot of discharge measurements and one-day-ahead forecast of daily maximum discharge.

4.4.3. LIMITATIONS

Even though it delivers impressive results, the model comes with some limitations. Firstly, it is a lumped model. These models often fail to account for the heterogeneity of a catchment. Secondly, it lacks an interception component; there is either net rainfall or net evaporation. Adding such a component is likely to improve model performance. Lastly, it is assumed 90% of the water quantity routes to the deeper soils and 10% ends up in the stream with less lag. This ratio might be calibrated too. Then again, the fact GR4J has only four calibration parameters is the model's essence.

It has to be noted that the direct forecasting method used here is actually invalid. GR4J is physically based and calibrating the model on tomorrow's discharge with today's discharge is physically impossible. The correct way would be to forecast rainfall and potential evaporation, and feed this forecast to the model. However, it is decided to use the same forecasting method to keep the comparison more or less fair, albeit a rough one.

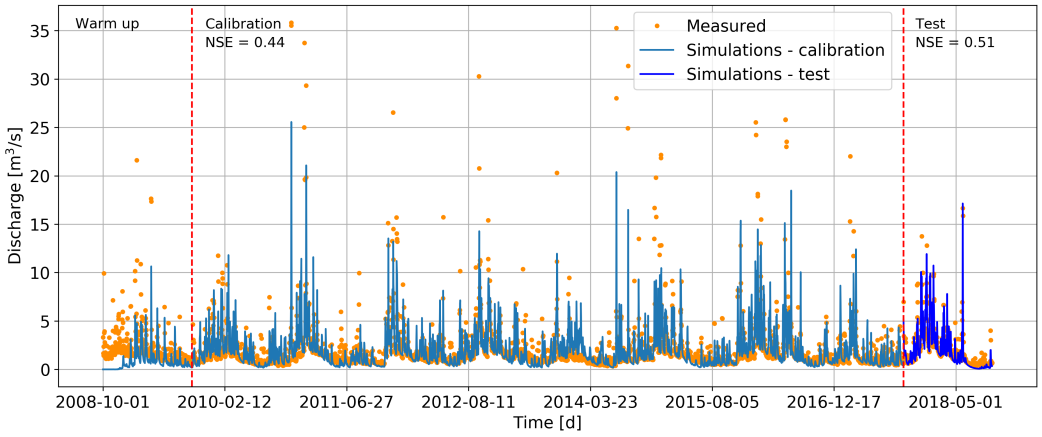

Figure 4.45: Hydrograph of measured discharge and one-day-ahead forecast of daily maximum discharge, including warm-up and calibration periods.

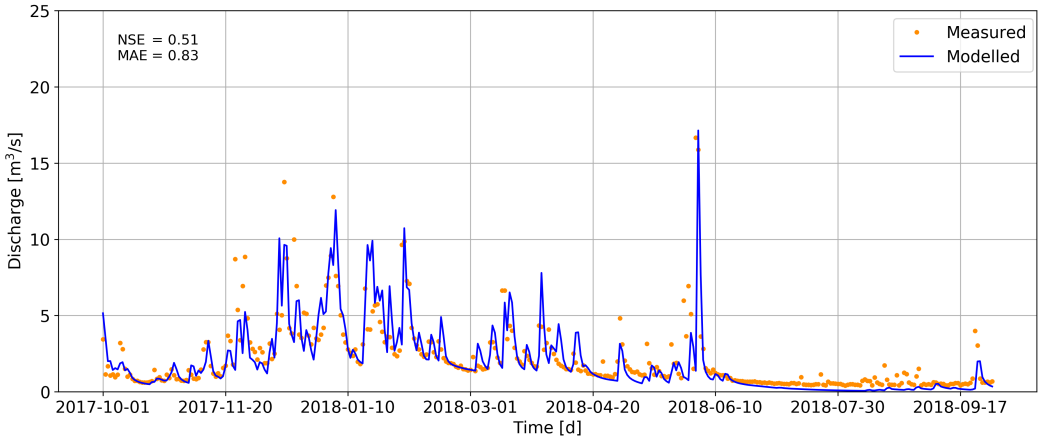

Figure 4.46: Hydrograph of measured discharge and one-day-ahead forecast of daily maximum discharge, test set only.

4.5. COMPARISON OF DATA-DRIVEN AND CONCEPTUAL

In this section a comparison is made. Firstly, the two machine learning models are compared to which performs best for simulating and forecasting discharge. Secondly, the two machine learning models and the conceptual model are compared on the daily resolution for simulating today's discharge as well as the one-day-ahead forecast. This is done for the Geul catchment. After the comparison, a step is taken back to look at the bigger picture regarding data-driven and conceptual modelling.

4.5.1. COMPARING GRADIENT BOOSTING AND DEEP LEARNING

With the results of both models applied to both catchments known, it is time to discuss which ML model performs best. The results are summarized in Table 4.8. The Geul catchment is discussed first. Starting with the simulation, NSE is higher for gradient boosting (0.71) compared to deep learning (0.69). MAE is higher for the former though (0.75 vs 0.70 m^3/s). With the differences so small, it is hard to tell which model performs better. However, deep learning gives better results when forecasting, having a higher NSE and lower MAE for all three forecasting periods. There is one exception and that is the three-days-ahead forecast, here NSE of the gradient boosting model is higher but so is the MAE compared to deep learning.
The results for the Rur are slightly different. Starting with the simulation, both ML models have an equal NSE (0.51) but MAE is lower for the deep learning model. The deep learning model has better metrics for the one-day-ahead and two-days-ahead forecast. Both NSE and MAE are better compared to the gradient boosted model. For the three-days-ahead forecast NSE is slightly higher for the gradient boosted model (0.33) compared to the deep learning one (0.30). However, MAE is slightly lower for the latter one. All in all, deep learning gives better results than gradient boosting.

Table 4.8: Metrics of the simulations and forecasts for the two catchments and the two machine learning methods. GB is gradient boosting and DL is deep learning.

	Geul catchment				Rur catchment			
	GB		DL		GB		DL	
	NSE	MAE	NSE	MAE	NSE	MAE	NSE	MAE
Simulation	0.71	0.75	0.69	0.70	0.51	5.63	0.51	5.49
1-day-ahead	0.57	0.81	0.58	0.78	0.42	6.02	0.45	5.83
2-days-ahead	0.41	0.99	0.46	0.88	0.35	6.30	0.37	6.19
3-days-ahead	0.37	1.05	0.29	0.98	0.33	6.42	0.30	6.36

4.5.2. DATA-DRIVEN AND CONCEPTUAL: DAILY DISCHARGE

By looking solely at the metrics (Table 4.9) there is no model outperforming the other two. Gradient boosting has the highest NSE (0.71) but deep learning has the lowest MAE (0.70 m^3/s). GR4J performs worse compared to the ML models; an NSE of 0.65 and a MAE of 0.83 m^3/s. When inspecting the hydrograph (Figure 4.47), at first sight, the models seem quite similar except at the largest peak and the peak at the end of the graph. At the former gradient boosting overestimates the peak whereas GR4J and deep learning underestimate it, with the absolute error being approximately 4-5 m^3/s for all three models. The latter is underestimated by GR4J but overestimated by the ML models. Overall, peak magnitude is estimated quite well. The deep learning model is best at modelling lower flows.

Table 4.9: Metrics of the simulations and forecasts of the three models for the Geul catchment.

	Simulation		Forecast	
Model	NSE	MAE	NSE	MAE
Gradient boosting	0.71	0.75	0.57	0.81
Deep learning	0.69	0.70	0.58	0.78
GR4J	0.65	0.81	0.51	0.83

4.5.3. DATA-DRIVEN AND CONCEPTUAL: FORECASTING DAILY DISCHARGE

Figure 4.48 shows the one-day-ahead forecast for the three models. The hydrograph of the ML models are slightly dampened compared to the daily simulation, except for the largest peak and the peak at the end of

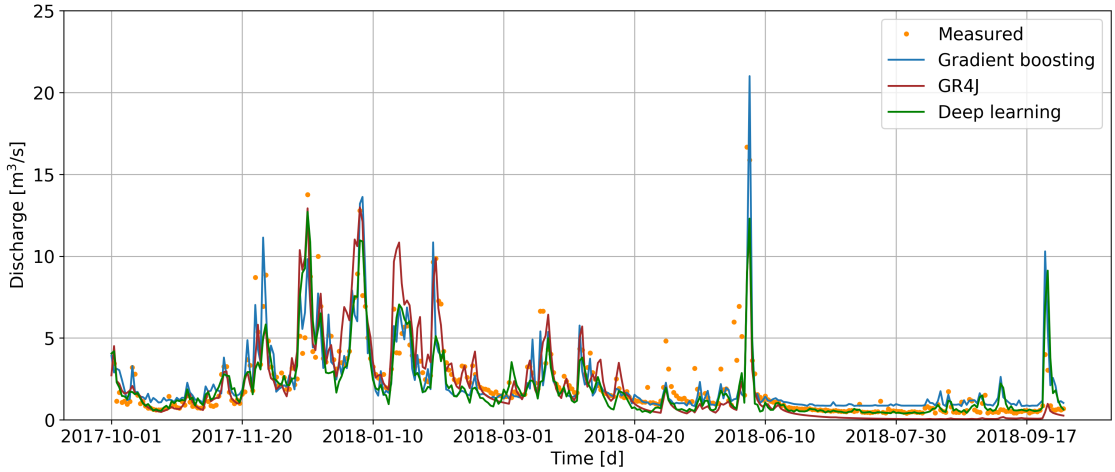

Figure 4.47: Hydrographs of measured discharge and maximum daily discharge simulated by gradient boosting, deep learning, and GR4J.

the series. These are overestimated, especially the latter point. The conceptual model remains the same in terms of magnitude. Peak timing is still accurate, but peak magnitude is off for the ML models. GR4J forecasts the largest peak at 17.0 m^3/s, which is close to the measured discharge of 16.7 m^3/s, it is forecast better than simulated. Looking at the metrics (Table 4.9), it shows that the deep learning model performs best with an NSE of 0.58 and MAE of m^3/s. Gradient boosting is slightly worse, with only a difference of 0.01 in NSE and 0.03 m^3/s in MAE. The conceptual model performs worst with an NSE of 0.51 and MAE of 0.83 m^3/s. All three models' performance classify as sufficient.

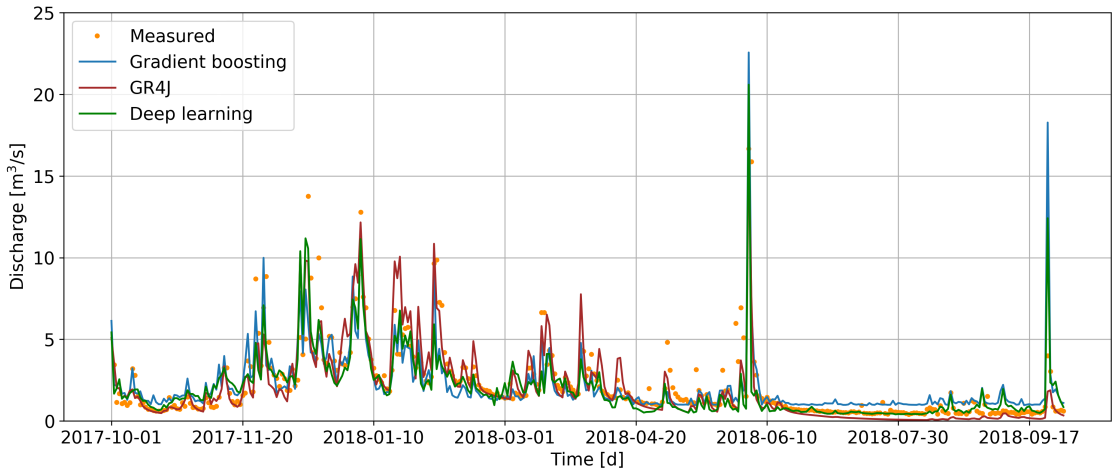

Figure 4.48: Hydrographs of measured discharge and maximum daily discharge forecast by XGBoost, deep learning, and GR4J.

4.5.4. THE BIGGER PICTURE

If one only looks at the metrics, the ML models outperform the conceptual model at simulating discharge (gradient boosting or deep learning performs best depending on the metric) and forecasting discharge (deep learning performs best). However, The differences are not that big so it is important to take a step back from the metrics and look at the bigger picture:

- GR4J only has two input variables: precipitation and potential evaporation. Both ML models require five input variables; precipitation, it's moving cumulative, soil moisture content, temperature and potential evaporation. The difference in metrics is not that big while one can reduce the amount of input variables from five to two.
- the conceptual model is transparent and one knows why a prediction is formed as it is based on physics.

For the ML models this is extremely hard to figure out, if not impossible. While it can be back tracked why a certain prediction is made, there is no physical meaning behind.
- The conceptual model needs to calibrate four parameters to which a physical meaning can be contributed. Hyperparameter tuning is mostly based on personal experience. An objective and computationally feasible way to tune hyperparameters is yet to be developed.

Combining the points above with the slight difference in metrics, the author would still favour the conceptual model.

However, it can be questioned whether conceptual models and data-driven model can be fairly compared in this study and overall. As mentioned in Section 4.4.3, forecasting is done using the direct method while the recursive method is appropriate for a conceptual or physically-based model. This can be solved by using the latter forecasting method for both models.
In addition, it is questionable whether a model with five input variables can be fairly compared to a model with only two input variables. If the degrees of freedom increase in a model, performance is likely to increase too. Therefore, a sort of Bayesian information criterion (BIC) [127] or Akaike information criterion (AIC) [128], that takes into account not just accuracy but also the amount of estimated parameters, might be more appropriate to use instead of or in addition to NSE and MAE. A problem with AIC is that it will not necessarily choose the correct model as sample size increases [129]. BIC is only valid when the sample size is much larger than the amount of trained parameters in a model [130]. This is not the case for the deep learning model. Therefore, a new criterion that takes into account not just accuracy but also model parsimony, much like the objective of XGBoost should be thought of. This would enable a much fairer and objective comparison of data-driven and conceptual or physically-based models.
Another issue is the difference between causation and correlation. Where conceptual and physically-based hydrologic models are based on causation, data-driven methods rely on correlation. Variables not related to hydrology might work too. For example, temperature has a seasonal cycle that could be mimicked by the amount of ice sales at a beach. Both will be high in the summer, and low in winter. Data-driven merely test whether input is correlated to output, and whether a causal link might exist. In that sense, purely data-driven models do not advance hydrologic sciences by suggesting new theories, it only points in a direction where a causal link might exist [131]. This can be helpful too; purely data-driven models can show how much discharge information can be extracted from datasets used in physically-based models [132].
In addition, data-driven models assume that all events have some kind of common behaviour. This is not necessarily true. High discharge can be caused by long stratiform precipitation or by a short convective precipitation event. Moreover, it is assumed that these common characteristics can be extracted by the data-driven model from past observations [131].
While the discussion up to now has been about the differences between knowledge-driven and data-driven, it is important to realize that one cannot function without the other. The former type needs data to calibrate parameters and to test hypotheses. The latter one performs best when process knowledge is involved when creating a data-driven model [133]. A way forward could be to use data-driven models to see how much signal is present in data. This signal can then be modelled using a knowledge-driven model.

5

CONCLUSIONS & FUTURE RESEARCH

This research aims to investigate whether river discharge can be forecast using machine learning methods, and whether these models outperform a conceptual model. To this end, a gradient boosting (XGBoost) model and a deep learning (LSTM) model were implemented for the Geul and Rur catchment. In addition, it was tested whether soil moisture content and NDVI improved model performance in addition to meteorological variables.

5.1. CONCLUSIONS

In order to come to the conclusion of this study, the research questions posed in the introduction are answered.

Sub question 1: What is the performance for different time horizon forecasts?
For the Geul when using gradient boosting or deep learning, NSE and MAE worsen as time horizon increases. Looking purely at NSE, the one-day-ahead forecast of both ML models is above 0.50, indicating model performance is sufficient. For the Rur this not the case. NSE is 0.42 and 0.45 for gradient boosting and deep learning, respectively. However, simulating daily maximum discharge is barely sufficient with an NSE of 0.51. While peak timing of the forecast is still accurate, magnitude is underestimated.

Sub question 2: Which type of ML model is best suited for forecasting discharge?
To answer this, only daily simulations and forecasts are taken into consideration. Deep learning proved to be best suited to forecast discharge for both catchments. Gradient boosting only outperforms deep learning when looking at NSE of the simulation of the Geul catchment and the three-days-ahead forecast of the Rur catchment. In addition, training a deep learning model on the natural logarithm of the discharge, and transforming it back by taking the natural exponent might increase performance. This is not the case for gradient boosting.

Sub question 3: Do soil moisture content and NDVI improve model performance?
To test this both variables are separately added to the base model consisting of precipitation and its five-day moving cumulative. Meteorological variables are added too to make a comparison. Soil moisture content improves model performance equally or more compared to meteorological variables. NDVI, in general, does not increase performance more compared to the other variables. It can be concluded that NDVI does increase performance, but compared to other variables it is better left out.

Sub question 4: Do the machine learning models outperform a conceptual model?
To answer this question, the conceptual GR4J model was implemented and compared to gradient boosting and deep learning. The gradient boosted model has the highest NSE when looking at modelling today's discharge. Deep learning has the lowest MAE. In the case of forecasting one day ahead, deep learning has both the highest NSE and lowest MAE. Looking purely at the metrics, these ML models do outperform the conceptual model. However, this is not as obvious from a holistic point of view. The conceptual model has fewer parameters to calibrate, is transparent and has only two input variables, whereas the ML models have many parameters to train, are difficult to physically interpret, and have four or five input variables.

Main research question: Is it possible to forecast discharge of the Rur and Geul river using a LSTM or GBDT model, and if so, to what extent?
Yes it is possible, but NSE and MAE worsen rather quickly as forecast time horizon increases. This might have to do with the forecasting method employed. It is likely that performance will increase when input variables are forecast, as simulating today's discharge is, by far, the most accurate model. Looking purely at the metrics, forecasting more than one day ahead gives insufficient model performance. While peak timing is still accurate, magnitude is underestimated. If not only metrics are taken into account, the conceptual GR4J model outperforms the two ML models. However, machine learning is still in its infancy and the potential is not yet fully exploited.

5.2. FUTURE RESEARCH

With the limitations already given, some directions and improvements for further research are given here.

- Higher resolution, both spatially and temporary, data generally increases performance of data-driven models. Especially for NDVI and SMC, a higher temporal resolution might improve performance of the hourly model. Now a single value is taken for the day. Moreover, the mores samples available to a machine learning model for training, the better it can learn complex patterns.
  Higher spatial resolution on precipitation data gives more information on the precipitation type (convective or frontal) and the distance to nearest drain. For example, the KNMI has precipitation data on a 1x1 km grid available for the Netherlands.
  Currently, ERA5 Reanalysis is used for the meteorological variables of the Rur catchment and precipitation for both catchments. An improvement would be to use the ERA5 Land data set with a resolution of 9x9 km versus the 31x31 km that is used in this study. One grid cell is larger than the Geul catchment so measured precipitation might have occurred outside the catchment, resulting in false peaks. In addition, for the Geul the weather station at Maastricht is used which is outside the Geul catchment. It is preferred if the station is located in the catchment.
- Instead of NDVI a variable that represents vegetation in and on the perimeter of creeks might be interesting. For example, percentage of creek length mowed. This might further improve the timing of peaks on a hourly time scale. On a daily time scale this effect is unlikely of much importance.
- Forecasting is now done by shifting the output series back in time. Another method would be to forecast the input variables using weather models. Especially forecast precipitation would be helpful as this variable is the main forcer of the models.
- A major issue with data-driven models is their 'black-boxness' regarding physical interpretation of what is happening in the model. Future research can focus on interpretation of such models. If people do not trust a model, they will probably not use it. There are frameworks that try to interpret ML predictions to this end. Two examples are DeepShap [134] and LIME [135]. Data-driven models are more likely to be accepted in water resources management if scientists and users in general know what is going on inside and how predictions are formed.
- For both algorithms and catchments, it has been attempted only once to find out which variables are most important. A recommendation would be to repeat the process of finding the best variables for each time scale.
- For the deep learning model, one network configuration is used for all applications to save time. It is better to tune the configuration to the specific application.
- In order to better model the magnitude of peaks, discharge could be separated into peak and base flow. For each component a separate model can be made, and the final prediction would be the sum of the two.
- Groundwater level near the rivers might improve model performance, especially for the Geul as base flow mainly consists of groundwater.
- Use more metrics to evaluate the models. Only using NSE and MAE is insufficient. Having metrics to describe how well low flows and high flows are simulated makes it easier to compare models, and makes measuring model performance more objective. Some additional metrics might be the NSE of the logarithm of the discharges and mean absolute percentage error.

BIBLIOGRAPHY

[1] C. L. M. S. . European Union, *Corine land cover 2018,* .

[2] A. Bhande, *What is underfitting and overfitting in machine learning and how to deal with it.* .

[3] T. Chen and C. Guestrin, *Xgboost: A scalable tree boosting system,* in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16 (ACM, New York, NY, USA, 2016) pp. 785–794.

[4] F. Chollet, *Deep learning with python* (Shelter Island, NY: Manning Publications, 2018).

[5] C. Perrin, C. Michel, and V. Andréassian, *Improvement of a parsimonious model for streamflow simulation,* Journal of hydrology **279**, 275 (2003).

[6] D. N. Moriasi, J. G. Arnold, M. W. Van Liew, R. L. Bingner, R. D. Harmel, and T. L. Veith, *Model evaluation guidelines for systematic quantification of accuracy in watershed simulations,* Transactions of the ASABE **50**, 885 (2007).

[7] B. van den Hurk, P. Siegmund, A. K. T. (Eds), J. Attema, A. Bakker, J. Beersma, J. Bessembinder, R. Boers, T. Brandsma, H. van den Brink, S. Drijfhout, H. Eskes, R. Haarsma, W. Hazeleger, R. Jilderda, C. Katsman, G. Lenderink, J. Loriaux, E. van Meijgaard, T. van Noije, G. J. van Oldenborgh, F. Selten, P. Siebesma, A. Sterl, H. de Vries, M. van Weele, R. de Winter, and G.-J. van Zadelhoff, *KNMI'14: Climate Change scenarios for the 21st Century – A Netherlands perspective*, Tech. Rep. (Royal Netherlands Meteorological Institute, KNMI, De Bilt, The Netherlands, 2014).

[8] L. De Gregorio, M. Callegari, P. Mazzoli, S. Bagli, D. Broccoli, A. Pistocchi, and C. Notarnicola, *Operational river discharge forecasting with support vector regression technique applied to alpine catchments: Results, advantages, limits and lesson learned,* Water Resources Management **32**, 229 (2018).

[9] J. Sitterson, C. Knightes, R. Parmar, K. Wolfe, M. Muche, and B. Avant, *An Overview of Rainfall-Runoff Model Types*, Tech. Rep. (United States Environmental Protection Agency, Office of Research and Development National Exposure Research Laboratory, 2017).

[10] E. Engman and R. Guerney, *Remote sensing in hydrology* (Springer Netherlands, New York, 1991).

[11] D. Maidment, *Arc Hydro: GIS for Water Resources* (ESRI Press, Redlands CA, 2002).

[12] S. Liangs, X. Li, and X. Xie, *Land surface observation, modeling and data assimilation* (World Scientific, Singapore, 2013).

[13] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, *In-datacenter performance analysis of a tensor processing unit,* SIGARCH Comput. Archit. News **45**, 1 (2017).

[14] J. Osborne, *Google's tensor processing unit explained: this is what the future of computing looks like,* .

[15] *Data-driven,* (2020).

[16] A. Oberoi, *9 machine learning examples from day-to-day life,* .

[17] Y. Han, J. Wu, B. Zhai, Y. Pan, G. Huang, L. Wu, and W. Zeng, *Coupling a bat algorithm with xgboost to estimate reference evapotranspiration in the arid and semiarid regions of china,* Advances in Meteorology **2019** (2019).

[18] L. Wu and J. Fan, *Comparison of neuron-based, kernel-based, tree-based and curve-based machine learning models for predicting daily reference evapotranspiration,* PloS one **14** (2019).

[19] H. Tyralis, G. Papacharalampous, and A. Langousis, *Super learning for daily streamflow forecasting: Large-scale demonstration and comparison with multiple machine learning algorithms,* arXiv preprint arXiv:1909.04131 (2019).

[20] B. Snider and E. A. McBean, *Improving urban water security through pipe-break prediction models: Machine learning or survival analysis,* Journal of Environmental Engineering **146**, 04019129 (2020).

[21] K. Loggenberg, A. Strever, B. Greyling, and N. Poona, *Modelling water stress in a shiraz vineyard using hyperspectral imaging and machine learning,* Remote Sensing **10**, 202 (2018).

[22] S. Cheng, S. Zhang, L. Li, and D. Zhang, *Water quality monitoring method based on tld 3d fish tracking and xgboost,* Mathematical Problems in Engineering **2018** (2018).

[23] D. De Clercq, K. Smith, B. Chou, A. Gonzalez, R. Kothapalle, C. Li, X. Dong, S. Liu, and Z. Wen, *Identification of urban drinking water supply patterns across 627 cities in china based on supervised and unsupervised statistical learning,* Journal of environmental management **223**, 658 (2018).

[24] Y. Lee, D. Han, M.-H. Ahn, J. Im, and S. J. Lee, *Retrieval of total precipitable water from himawari-8 ahi data: a comparison of random forest, extreme gradient boosting, and deep neural network,* Remote Sensing **11**, 1741 (2019).

[25] H. Zhang, Q. Yang, J. Shao, and G. Wang, *Dynamic streamflow simulation via online gradient-boosted regression tree,* Journal of Hydrologic Engineering **24**, 04019041 (2019).

[26] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction* (Springer Science & Business Media, 2009).

[27] C. Robert, *Machine learning, a probabilistic perspective* (Taylor & Francis, 2014).

[28] W.-Y. Loh and Y.-S. Shih, *Split selection methods for classification trees,* Statistica sinica , 815 (1997).

[29] C. Strobl, A.-L. Boulesteix, and T. Augustin, *Unbiased split selection for classification trees based on the gini index,* Computational Statistics & Data Analysis **52**, 483 (2007).

[30] M. Kuhn and K. Johnson, *Applied predictive modeling,* Vol. 26 (Springer, 2013).

[31] J. Zhou, Y. Wang, F. Xiao, Y. Wang, and L. Sun, *Water quality prediction method based on igra and lstm,* Water **10**, 1148 (2018).

[32] Z. Li, F. Peng, B. Niu, G. Li, J. Wu, and Z. Miao, *Water quality prediction model combining sparse auto-encoder and lstm network,* IFAC-PapersOnLine **51**, 831 (2018).

[33] P. Liu, J. Wang, A. K. Sangaiah, Y. Xie, and X. Yin, *Analysis and prediction of water quality using lstm deep neural networks in iot environment,* Sustainability **11**, 2058 (2019).

[34] F. Kratzert, D. Klotz, C. Brenner, K. Schulz, and M. Herrnegger, *Rainfall-runoff modelling using long short-term memory (lstm) networks,* Hydrology and Earth System Sciences **22**, 6005 (2018).

[35] L. Ni, D. Wang, V. P. Singh, J. Wu, Y. Wang, Y. Tao, and J. Zhang, *Streamflow and rainfall forecasting by two long short-term memory-based models,* Journal of Hydrology , 124296 (2019).

[36] X.-H. Le, H. V. Ho, G. Lee, and S. Jung, *Application of long short-term memory (lstm) neural network for flood forecasting,* Water **11**, 1387 (2019).

[37] R. Hu, F. Fang, C. Pain, and I. Navon, *Rapid spatio-temporal flood prediction and uncertainty quantification using a deep learning method,* Journal of Hydrology **575**, 911 (2019).

[38] T. Yang, F. Sun, P. Gentine, W. Liu, H. Wang, J. Yin, M. Du, and C. Liu, *Evaluation and machine learning improvement of global hydrological model-based flood simulations,* Environmental Research Letters **14**, 114027 (2019).

[39] Y. Qi, Z. Zhou, L. Yang, Y. Quan, and Q. Miao, *A decomposition-ensemble learning model based on lstm neural network for daily reservoir inflow forecasting,* Water Resources Management **33**, 4123 (2019).

[40] S. Yang, D. Yang, J. Chen, and B. Zhao, *Real-time reservoir operation using recurrent neural networks and inflow forecast from a distributed hydrological model,* Journal of Hydrology **579**, 124229 (2019).

[41] D. Tran Anh, S. P. Van, T. D. Dang, and L. P. Hoang, *Downscaling rainfall using deep learning long short-term memory and feedforward neural network,* International Journal of Climatology **39**, 4170 (2019).

[42] A. Akbari Asanjan, T. Yang, K. Hsu, S. Sorooshian, J. Lin, and Q. Peng, *Short-term precipitation forecast based on the persiann system and lstm recurrent neural networks,* Journal of Geophysical Research: Atmospheres **123**, 12 (2018).

[43] D. Kumar, A. Singh, P. Samui, and R. K. Jha, *Forecasting monthly precipitation using sequential modelling,* Hydrological sciences journal **64**, 690 (2019).

[44] J. Zhang, Y. Zhu, X. Zhang, M. Ye, and J. Yang, *Developing a long short-term memory (lstm) based model for predicting water table depth in agricultural areas,* Journal of hydrology **561**, 918 (2018).

[45] B. Hrnjica and O. Bonacci, *Lake level prediction using feed forward and recurrent neural networks,* Water Resources Management **33**, 2471 (2019).

[46] D. Zhang, N. Martinez, G. Lindholm, and H. Ratnaweera, *Manage sewer in-line storage control using hydraulic model and recurrent neural network,* Water resources management **32**, 2079 (2018).

[47] D. Zhang, G. Lindholm, and H. Ratnaweera, *Use long short-term memory to enhance internet of things for combined sewer overflow monitoring,* Journal of Hydrology **556**, 409 (2018).

[48] A. T. C. on Application of Artificial Neural Networks in Hydrology, *Artificial neural networks in hydrology. ii: Hydrologic applications,* Journal of Hydrologic Engineering **5**, 124 (2000).

[49] *Website of waterschap limburg (dutch only),* .

[50] W.-C. Wang, K.-W. Chau, C.-T. Cheng, and L. Qiu, *A comparison of performance of several artificial intelligence methods for forecasting monthly discharge time series,* Journal of Hydrology **374**, 294 (2009).

[51] Z. Liu, P. Zhou, G. Chen, and L. Guo, *Evaluating a coupled discrete wavelet transform and support vector regression for daily and monthly streamflow forecasting,* Journal of Hydrology **519**, 2822 (2014).

[52] V. Nourani, A. H. Baghanam, J. Adamowski, and O. Kisi, *Applications of hybrid wavelet–artificial intelligence models in hydrology: A review,* Journal of Hydrology **514**, 358 (2014).

[53] H. Asadi, K. Shahedi, B. Jarihani, and R. C. Sidle, *Rainfall-runoff modelling using hydrological connectivity index and artificial neural network approach,* Water **11** (2019), 10.3390/w11020212.

[54] T. Ochsner, M. Cosh, R. Cuenca, W. Dorigo, C. Draper, and Y. H. et al., *State of the art in large-scale soil moisture monitoring,* Soil Science Society of America Journal **77** (2013).

[55] L. Brocca, T. Moramarco, F. Melone, W. Wagner, S. Hasenauer, and S. Hahn, *Assimilation of surface-and root-zone ascat soil moisture products into rainfall–runoff modeling,* IEEE Transactions on Geoscience and Remote Sensing **50**, 2542 (2011).

[56] C. Hu, Q. Wu, H. Li, S. Jian, N. Li, and Z. Lou, *Deep learning with a long short-term memory networks approach for rainfall-runoff simulation,* Water **10** (2018), 10.3390/w10111543.

[57] S. Dautrebande, J. Leenaars, J. Schmitz, and E. Vanthournout, *Pilot project for the definition of environment-friendly measures to reduce the risk for flash floods in the Geul River catchment*, Tech. Rep. (Technum, University of Liège and CSO Environmental Consultancy, 2000).

[58] Unknown, *Knmi: Extreme neerslag [nl],* ().

[59] P. D. laat and M. Chu-Agor, *Geen toename piekafvoer van de geul,* H2O , 25 (2003).

[60] M. Armstrong, *Geostatistics: Proceedings of the Third International Geostatistics Congress September 5–9, 1988, Avignon, France*, Vol. 4 (Springer Science & Business Media, 2013).

[61] J. De Moor, C. Kasse, R. Van Balen, J. Vandenberghe, and J. Wallinga, *Human and climate impact on catchment development during the holocene—geul river, the netherlands,* Geomorphology **98**, 316 (2008).

[62] R. Verborg and J. Tiems, *Modderstromen in zuid-limburg na hevige regenval [nl],* .

[63] GSK3B, *Gewässerstationierungskarte des landes nordrhein-westfalen,* .

[64] H. Bogena, M. Herbst, J. Hake, K. Kunkel, C. Montzka, T. Putz, H. Vereecken, and F. Wendland, *Mosyrur - water balance analysis in the rur basin,* Environment **52** (2005).

[65] M. Kufled, J. Lange, and B. Hausmann, *Das Einzugsgebiet der Rur; Ergebnisbericht der im Rahmen des AMICE–Projekts durchgeführten Literaturrecherche*, Tech. Rep. (AMICE; Adaptation of the Meuse to the Impacts of Climate Evolutions, 2010).

[66] C. C. C. S. C. (2017), *Era5: Fifth generation of ecmwf atmospheric reanalyses of the global climate,* .

[67] Unknown, *Knmi: uitleg over automatische weerstations [nl],* ().

[68] R. B. Myneni, F. G. Hall, P. J. Sellers, and A. L. Marshak, *The interpretation of spectral vegetation indexes,* IEEE Transactions on Geoscience and Remote Sensing **33**, 481 (1995).

[69] J. Rouse Jr, R. Haas, J. Schell, and D. Deering, *Monitoring vegetation systems in the great plains with erts,* in 3rd ERTS Symp., NASA SP-351 (U.S. Gov. Printing Office, Washington, DC, 1973) pp. 309–317.

[70] *Sentinel hub - ndvi (normalized vegetation difference index),* .

[71] N. Nikora, V. Nikora, and T. O'Donoghue, *Velocity profiles in vegetated open-channel flows: combined effects of multiple mechanisms,* Journal of Hydraulic Engineering **139**, 1021 (2013).

[72] E. Vermote, *Noaa climate data record (cdr) of avhrr normalized difference vegetation index (ndvi), version 5. subset: 01-10-2008 to 30-09-2018,* .

[73] A. Robock, *Hydrology, floods and droughts | soil moisture,* in *Encyclopedia of Atmospheric Sciences (Second Edition)*, edited by G. R. North, J. Pyle, and F. Zhang (Academic Press, Oxford, 2015) second edition ed., pp. 232 – 239.

[74] W. T. Crow, R. Bindlish, and T. J. Jackson, *The added value of spaceborne passive microwave soil moisture retrievals for forecasting rainfall-runoff partitioning,* Geophysical Research Letters **32** (2005), 10.1029/2005GL023543, https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2005GL023543 .

[75] L. Berthet, V. Andréassian, C. Perrin, and P. Javelle, *How crucial is it to account for the antecedent moisture conditions in flood forecasting? comparison of event-based and continuous approaches on 178 catchments,* Hydrology and Earth System Sciences **13**, 819 (2009).

[76] W. Dorigo, W. Wagner, C. Albergel, F. Albrecht, G. Balsamo, L. Brocca, D. Chung, M. Ertl, M. Forkel, A. Gruber, *et al.*, *Esa cci soil moisture for improved earth system understanding: State-of-the art and future directions,* Remote Sensing of Environment **203**, 185 (2017).

[77] A. Gruber, T. Scanlon, R. v. d. Schalie, W. Wagner, and W. Dorigo, *Evolution of the esa cci soil moisture climate data records and their underlying merging methodology,* Earth System Science Data **11**, 717 (2019).

[78] A. Gruber, W. A. Dorigo, W. Crow, and W. Wagner, *Triple collocation-based merging of satellite soil moisture retrievals,* IEEE Transactions on Geoscience and Remote Sensing **55**, 6780 (2017).

[79] R. Kidd and E. Haas, *ESA Climate Change Initiative PlusSoil Moisture; Soil Moisture ECV Product User Guide (PUG) Revision 3 D3.3.1 Version 4.5*, Tech. Rep. (European Space Agency (ESA), 2019).

[80] *Penman-Monteith referentieverdamping*, Tech. Rep. (Amersfoort: Stowa, 2010).

[81] Unknown, *Handboek Waarnemingen*, Tech. Rep. (Royal Netherlands Meteorological Institute (KNMI), 2001).

[82] L. E. Back and C. S. Bretherton, *The relationship between wind speed and precipitation in the pacific itcz,* Journal of climate **18**, 4317 (2005).

[83] G. Rossum, *Python Reference Manual*, Tech. Rep. (CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, The Netherlands, 1995).

[84] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, *Scikit-learn: Machine learning in python,* Journal of machine learning research **12**, 2825 (2011).

[85] W. McKinney *et al.*, *Data structures for statistical computing in python,* in *Proceedings of the 9th Python in Science Conference*, Vol. 445 (Austin, TX, 2010) pp. 51–56.

[86] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, *The numpy array: a structure for efficient numerical computation,* Computing in Science & Engineering **13**, 22 (2011).

[87] S. Hoyer and J. Hamman, *xarray: N-D labeled arrays and datasets in Python,* Journal of Open Research Software **5** (2017), 10.5334/jors.148.

[88] F. Chollet, *Keras,* `https://github.com/fchollet/keras` (2015).

[89] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems,* (2015), software available from tensorflow.org.

[90] J. D. Hunter, *Matplotlib: A 2d graphics environment,* Computing in science & engineering **9**, 90 (2007).

[91] Y. Ji, J. Hao, N. Reyhani, and A. Lendasse, *Direct and recursive prediction of time series using mutual information selection,* in *International Work-Conference on Artificial Neural Networks* (Springer, 2005) pp. 1010–1017.

[92] A. Sorjamaa, J. Hao, N. Reyhani, Y. Ji, and A. Lendasse, *Methodology for long-term prediction of time series,* Neurocomputing **70**, 2861 (2007).

[93] S. B. Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa, *A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition,* Expert systems with applications **39**, 7067 (2012).

[94] J. H. Friedman, *Greedy function approximation: a gradient boosting machine,* Annals of statistics , 1189 (2001).

[95] M. Mishra, *Regularization: An important concept in machine learning,* .

[96] J. Brownlee, *What is the difference between a parameter and a hyperparameter?* .

[97] H. Hoos and K. Leyton-Brown, *An efficient approach for assessing hyperparameter importance,* in *International conference on machine learning* (2014) pp. 754–762.

[98] N. Reimers and I. Gurevych, *Optimal hyperparameters for deep lstm-networks for sequence labeling tasks,* arXiv preprint arXiv:1707.06799 (2017).

[99] J. Bergstra and Y. Bengio, *Random search for hyper-parameter optimization,* Journal of Machine Learning Research **13**, 281 (2012).

[100] J. Snoek, H. Larochelle, and R. P. Adams, *Practical bayesian optimization of machine learning algorithms,* in *Advances in neural information processing systems* (2012) pp. 2951–2959.

[101] F. Hutter, J. Lücke, and L. Schmidt-Thieme, *Beyond manual tuning of hyperparameters,* KI-Künstliche Intelligenz **29**, 329 (2015).

[102] X. developers, *Xgboost parameters,* .

[103] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization,* arXiv preprint arXiv:1412.6980 (2014).

[104] P. Skalski, *How to train neural network faster with optimizers?* .

[105] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks,* in *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010) pp. 249–256.

[106] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, *et al.*, *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies,* (2001).

[107] S. Hochreiter and J. Schmidhuber, *Long short-term memory,* Neural computation **9**, 1735 (1997).

[108] D. T. Larose, *Data mining and predictive analytics* (John Wiley & Sons, 2015) Chap. 2.

[109] Y. Gal and Z. Ghahramani, *A theoretically grounded application of dropout in recurrent neural networks,* in *Advances in neural information processing systems* (2016) pp. 1019–1027.

[110] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting,* The journal of machine learning research **15**, 1929 (2014).

[111] MissingLink, *Neural network concepts: 7 types of neural network activation functions: How to choose?* .

[112] naive, *How to select number of hidden layers and number of memory cells in an lstm?* .

[113] Jinglesting, *Recurrent neural net (lstm) batch size and input,* .

[114] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, *Lstm: A search space odyssey,* IEEE transactions on neural networks and learning systems **28**, 2222 (2016).

[115] Y. Gal and Z. Ghahramani, *Dropout as a bayesian approximation: Representing model uncertainty in deep learning,* in *international conference on machine learning* (2016) pp. 1050–1059.

[116] J. Klaas, *Machine learning for finance: principles and practice for financial insiders* (Packt Publishing, 2019).

[117] Q. Duan, F. Pappenberger, A. Wood, H. L. Cloke, and J. Schaake, *Handbook of Hydrometeorological Ensemble Forecasting* (Springer, 2019).

[118] D. Harlan, M. Wangsadipura, and C. M. Munajat, *Rainfall-runoff modeling of citarum hulu river basin by using gr4j,* in *Proceedings of the world congress on engineering,* Vol. 2 (2010) pp. 4–8.

[119] A. Kunnath-Poovakka and T. Eldho, *A comparative study of conceptual rainfall-runoff models gr4j, awbm and sacramento at catchments in the upper godavari river basin, india,* Journal of Earth System Science **128**, 33 (2019).

[120] K. B. Kim, H.-H. Kwon, and D. Han, *Exploration of warm-up period in conceptual hydrological modelling,* Journal of Hydrology **556**, 194 (2018).

[121] P. Krause, D. Boyle, and F. Bäse, *Comparison of different efficiency criteria for hydrological model assessment,* (2005).

[122] J. Nash and J. Sutcliffe, *River flow forecasting through conceptual models part i — a discussion of principles,* Journal of Hydrology **10**, 282 (1970).

[123] B. P. Wilcox, W. J. Rawls, D. L. Brakensiek, and J. R. Wight, *Predicting runoff from rangeland catchments: A comparison of two models,* Water Resources Research **26**, 2401 (1990), https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/WR026i010p02401 .

[124] C. W. Dawson, R. J. Abrahart, and L. M. See, *Hydrotest: a web-based toolbox of evaluation metrics for the standardised assessment of hydrological forecasts,* Environmental Modelling & Software **22**, 1034 (2007).

[125] L. Boytsov, *Is it possible to have a higher train error than a test error in machine learning?* .

[126] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling,* arXiv preprint arXiv:1412.3555 (2014).

[127] G. Schwarz *et al.*, *Estimating the dimension of a model,* The annals of statistics **6**, 461 (1978).

[128] H. Akaike, *Information theory and an extension of the maximum likelihood principle,* in *Selected papers of hirotugu akaike* (Springer, 1998) pp. 199–213.

[129] L. Marshall, D. Nott, and A. Sharma, *Hydrological model selection: A bayesian alternative,* Water resources research **41** (2005).

[130] C. Giraud, *Introduction to high-dimensional statistics* (Chapman and Hall/CRC, 2014).

[131] J. A. Cunge, *Of data and models,* Journal of Hydroinformatics **5**, 75 (2003).

[132] M. Gauch, J. Mai, S. Gharari, and J. Lin, *Data-driven vs. physically-based streamflow prediction models,* in *Proceedings of 9th International Workshop on Climate Informatics* (2019).

[133] H. Savenije and N. de Vos, *CT4431 - Hydrological Modelling* (Delft University of Technology, 2009).

[134] S. M. Lundberg and S.-I. Lee, *A unified approach to interpreting model predictions,* in *Advances in neural information processing systems* (2017) pp. 4765–4774.

[135] M. T. Ribeiro, S. Singh, and C. Guestrin, *" why should i trust you?" explaining the predictions of any classifier,* in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (2016) pp. 1135–1144.

[136] K. Kawakami, *Supervised sequence labelling with recurrent neural networks,* Ph. D. thesis (2008).

A

APPENDIX A

A.1. VARIABLES HISTOGRAMS

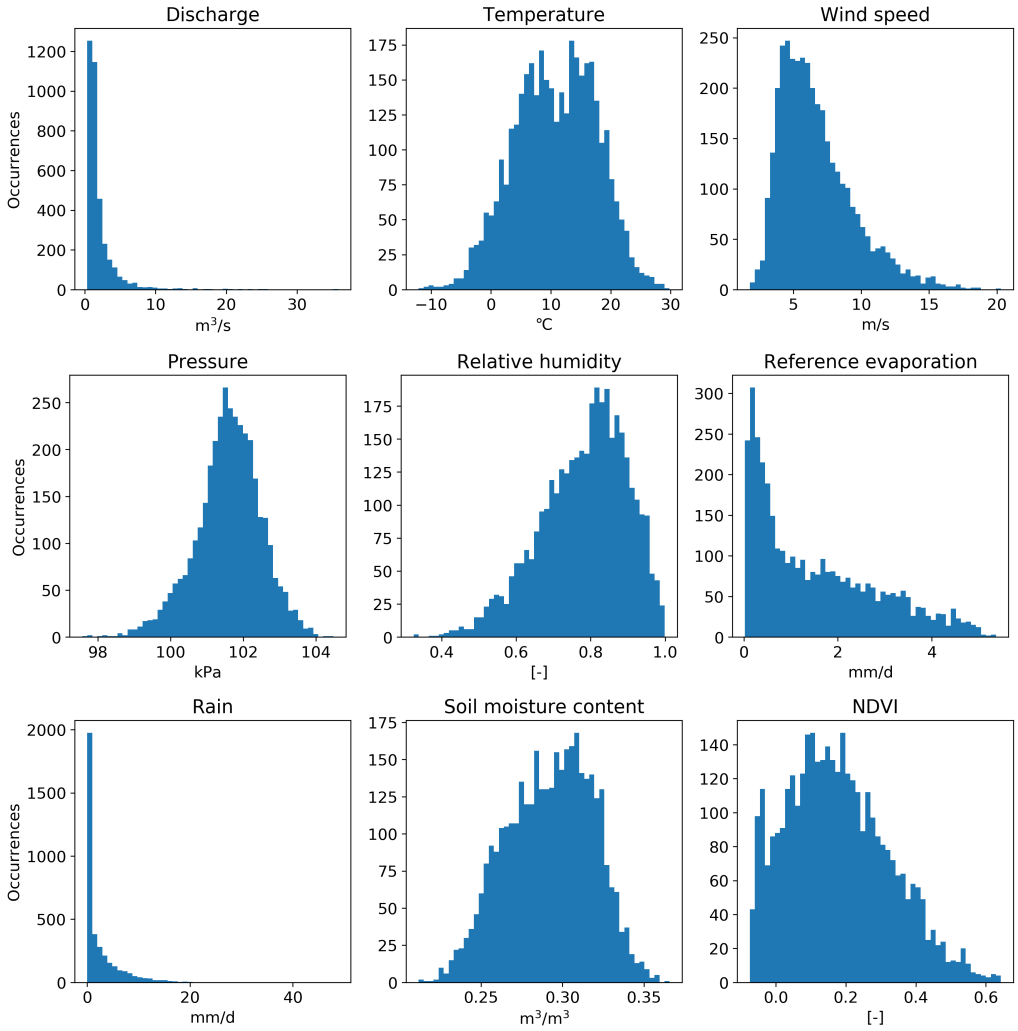

Figure A.1: Histogram of each variable used in this study. The variables are not normally distributed, save perhaps pressure.

A.2. LSTM

All the mathematics behind a LSTM cell is given here. It is split into two phases; forward pass and backward pass. In the following equations w_{ij} is the weight of the connection from i to j, input to unit j at time t is a_j^t, and activation of unit j at time t is b_j^t. For the input gate, output gate and forget gate the subscripts ι, ω and ϕ are used, respectively. The so-called peephole weights $w_{c\iota}$, $w_{c\omega}$ and $w_{c\phi}$ are the weights from cell c to input gate, output gate and forget gate, respectively. The state of cell c at time t is denoted by s_c^t. The activation function of the gates, cell input and cell output are denoted by f, g and h. The number of inputs, outputs and cells in a hidden layer are given by I, K, and H. Note that these equations are for just one cell. [136]

A.2.1. FORWARD PASS

Input gates

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} \tag{A.1}$$

$$b_\iota^t = f(a_\iota^t) \tag{A.2}$$

Forget gates

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1} \tag{A.3}$$

$$b_\phi^t = f(a_\phi^t) \tag{A.4}$$

Cells

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \tag{A.5}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t) \tag{A.6}$$

Output gates

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t \tag{A.7}$$

$$b_\omega^t = f(a_\omega^t) \tag{A.8}$$

Cell outputs

$$b_c^t = b_{c\omega}^t h(s_c^t) \tag{A.9}$$

A.2.2. BACKWARD PASS

$$\epsilon_c^t \equiv \frac{\partial \mathfrak{L}}{\partial b_c^t}, \qquad \epsilon_s^t \equiv \frac{\partial \mathfrak{L}}{\partial s_c^t} \tag{A.10}$$

Cell outputs

$$\epsilon_c^t = \sum_{k=1}^{K} w_{ck} \delta_k^t + \sum_{h=1}^{H} w_{ch} \delta_h^{t+1} \tag{A.11}$$

Output gates

$$\delta_\omega^t = f'(a_\omega^t) \sum_{c=1}^{C} h(s_c^t) \epsilon_c^t \tag{A.12}$$

States

$$\epsilon_s^t = b_\omega^t h'(s_s^t) \epsilon_c^t + b_\phi^{t+1} \epsilon_s^{t+1} + w_{cl} \delta_\iota^{t+1} + w_{c\phi} \delta_\phi^{t+1} + w_{cw} \delta_\omega^t \tag{A.13}$$

Cells

$$\delta_c^t = b_\iota^t g'(a_c^t) \epsilon_s^t \tag{A.14}$$

Forget gates

$$\delta_\phi^t = f'(a_\phi^t) \sum_{c=1}^{C} s_c^{t-1} \epsilon_s^t \tag{A.15}$$

Input gates

$$\delta_\iota^t = f'(a_\iota^t) \sum_{c=1}^{C} g'(a_c^t) \epsilon_s^t \tag{A.16}$$

A.3. XAVIER WEIGHT INITIALIZATION

The general idea is to initialize weights randomly from a standard normal distribution $\mathcal{N}(0,1)$. However, as it is preferable to keep variance of the input and output variables equal. Thus one needs to draw from a normal distribution with the same mean but a different variance. Firstly, let us calculate the the variance of the output y (Equation A.17).

$$var(y) = var(w_1x_1 + w_2x_2 + ... + w_nx_n + b) \tag{A.17}$$

where w_i are the weights, x_n the input values and b the bias. The variance of a separate term on the right hand side can be written as in Equation A.18 by using the product of independent variables.

$$var(w_ix_i) = E(x_i)^2 var(w_i) + E(w_i)^2 var(x_i) + var(w_i)var(x_i) \tag{A.18}$$

where E is the expectation of a variable. However, as the used distribution is assumed normal with a mean zero, this term vanishes, resulting in Equation A.19.

$$var(w_ix_i) = var(w_i)var(x_i) \tag{A.19}$$

Now, given that the bias is a constant and thus has zero variance, and substituting Equation A.19 in Equation A.17 results in Equation A.20.

$$var(y) = var(w_1x_1) + var(w_2x_2) + ... + var(w_nx_n) \tag{A.20}$$

As they are distributed identically, it can be rewritten to Equation A.21.

$$var(y) = N \cdot var(w_i) \cdot var(x_i) \tag{A.21}$$

From above equation it becomes clear that in order for the variance of x and y to be equal, the first term on the left should be equal to one. Rewriting this results in the final equation:

$$var(w_i) = \frac{1}{N} \tag{A.22}$$

where N is the mean of the amount of nodes in the input and output layer. The final weight distribution becomes $\mathcal{N}(0, var(w_i))$. [105]

B

Appendix B

B.1. Gradient Boosting: Geul

B.1.1. Results of all models leading up to the final model

Table B.1: Performance and hyperparameters of all models predicting today's discharge for the Geul. The best performing models of each iteration have been marked green. The best overall model is marked bold.

	NSE			MAE			Hyperparameters				
Model	Train	Val	Test	Train	Val	Test	Colsample	η	Max depth	n_est	Subsample
Baseline 1: rain, rainMA	0.39	0.23	0.44	1.09	1.15	1.15	0.822	0.0127	4	848	0.522
Baseline 1 + T	0.53	0.33	0.59	0.94	1.04	0.88	0.768	0.0237	4	1151	0.514
Baseline 1 + WS	0.47	0.28	0.50	1.05	1.13	1.10	0.758	0.0293	4	850	0.568
Baseline 1 + P	0.52	0.25	0.48	1.00	1.12	1.08	0.758	0.0293	4	850	0.568
Baseline 1 + RH	0.47	0.28	0.50	1.03	1.11	1.03	0.758	0.0293	5	850	0.568
Baseline 1 + ET	0.63	0.32	0.58	0.90	1.07	0.96	0.920	0.0294	5	1076	0.938
Baseline 1 + SMC	0.70	0.43	0.59	0.81	0.98	0.88	0.693	0.0261	5	1145	0.874
Baseline 1 + NDVI	0.49	0.29	0.49	1.01	1.10	1.06	0.758	0.0292	4	850	0.567
Baseline 2: rain, rainMA, T, SMC	0.68	0.46	0.66	0.8	0.94	0.79	0.973	0.0103	4	1046	0.975
Baseline 2 + WS	0.74	0.44	0.64	0.75	0.93	0.80	0.845	0.0111	5	1071	0.959
Baseline 2 + P	0.69	0.47	0.61	0.8	0.93	0.80	0.973	0.0103	4	1046	0.975
Baseline 2 + RH	0.73	0.44	0.66	0.75	0.94	0.78	0.958	0.0117	5	867	0.939
Baseline 2 + ET	0.75	0.45	0.70	0.75	0.93	0.77	0.898	0.0234	5	826	0.685
Baseline 2 + NDVI	0.68	0.45	0.63	0.8	0.93	0.80	0.958	0.0117	4	867	0.939
Baseline 3: rain, rainMA, T, SMC, ET	0.74	0.45	0.70	0.75	0.93	0.77	0.898	0.0234	5	826	0.685
Baseline 3 + WS	0.84	0.44	0.65	0.61	0.92	0.80	0.958	0.0186	7	1065	0.811
Baseline 3 + P	0.83	0.46	0.63	0.65	0.94	0.82	0.726	0.0279	6	873	0.894
Baseline 3 + RH	0.81	0.43	0.70	0.67	0.92	0.77	0.834	0.0294	6	807	0.900
Baseline 3 + NDVI	0.81	0.44	0.66	0.66	0.93	0.79	0.834	0.0294	6	807	0.900
Baseline 3 + Tt-1	0.85	0.44	0.68	0.59	0.92	0.75	0.958	0.0186	7	1065	0.811
Baseline 3 + Tt-2	0.77	0.45	0.67	0.73	0.93	0.75	0.898	0.0234	5	826	0.685
Baseline 3 + Tt-3	0.74	0.47	0.68	0.75	0.91	0.79	0.972	0.0103	4	1046	0.975
Baseline 3 + SMCt-1	0.82	0.44	0.69	0.66	0.92	0.77	0.897	0.0139	6	1188	0.923
Baseline 3 + SMCt-2	0.72	0.45	0.69	0.77	0.93	0.77	0.958	0.0117	4	867	0.939
Baseline 3 + SMCt-3	0.73	0.45	0.67	0.76	0.93	0.78	0.973	0.0103	4	1046	0.975
Baseline 3 + ETt-1	0.82	0.45	0.7	0.66	0.92	0.76	0.834	0.0294	6	807	0.900
Baseline 3 + ETt-2	**0.84**	**0.46**	**0.71**	**0.64**	**0.92**	**0.75**	**0.898**	**0.0139**	**6**	**1188**	**0.923**
Baseline 3 + ETt-3	0.89	0.44	0.71	0.54	0.93	0.75	0.976	0.0216	7	1137	0.953

B.1.2. DAILY FORECASTING

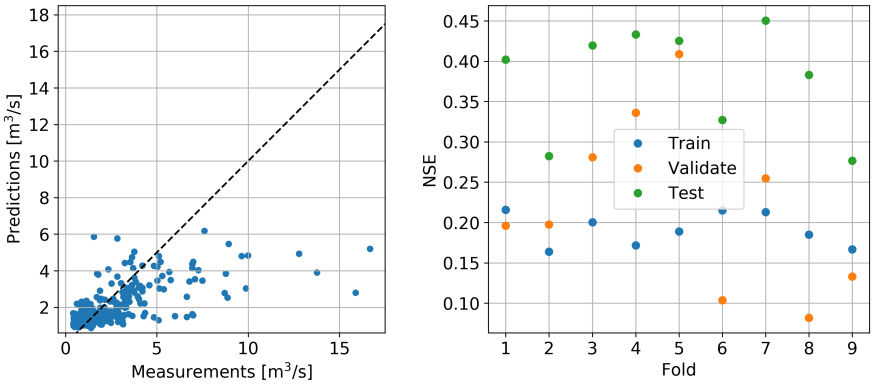

Figure B.1: 2-days-ahead forecast of daily maximum discharge using gradient boosting. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

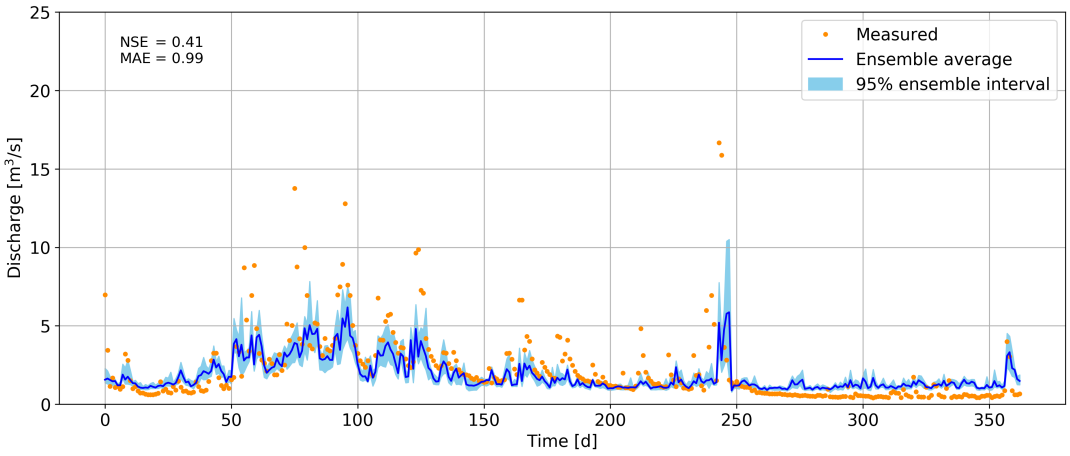

Figure B.2: Hydrograph of measured discharge and 2-days-ahead forecast of daily maximum discharge using gradient boosting.

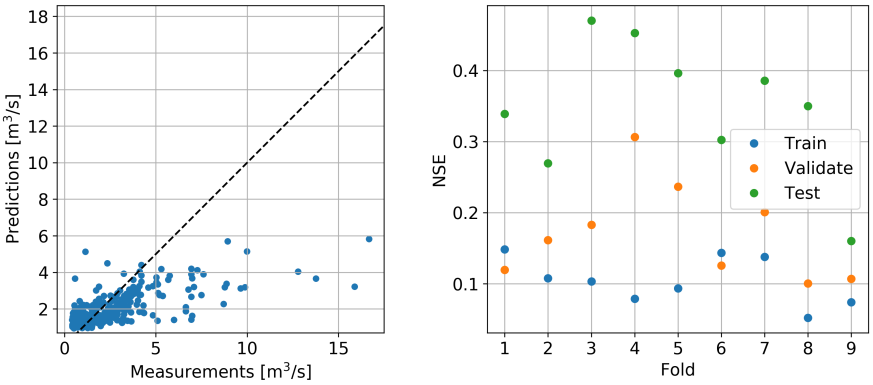

Figure B.3: 3-days-ahead forecast of daily maximum discharge using gradient boosting. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

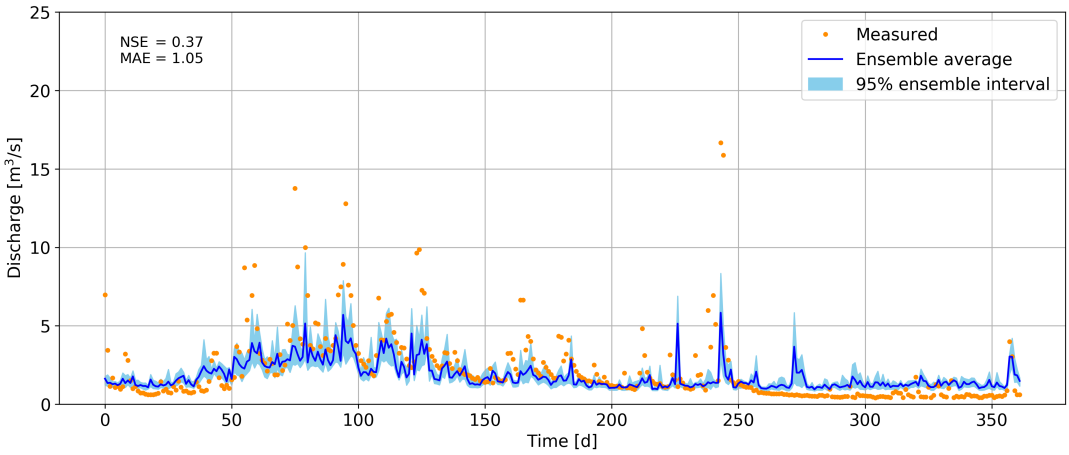

Figure B.4: Hydrograph of measured discharge and 3-days-ahead forecast of daily maximum discharge using gradient boosting.

B.2. GRADIENT BOOSTING: RUR

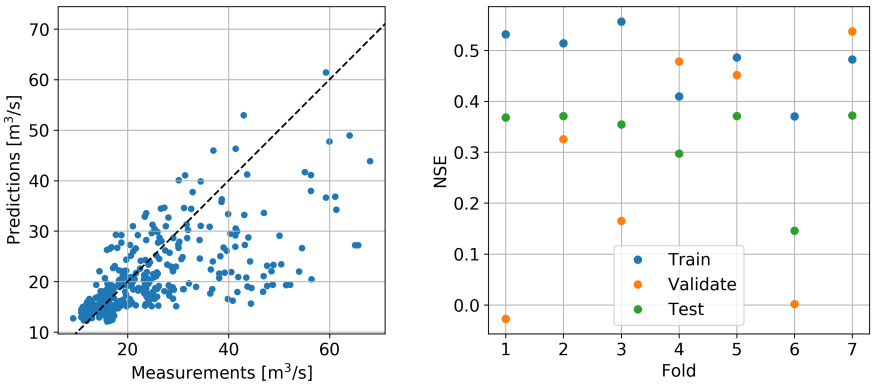

Figure B.5: 2-days-ahead forecast of daily maximum discharge using gradient boosting. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

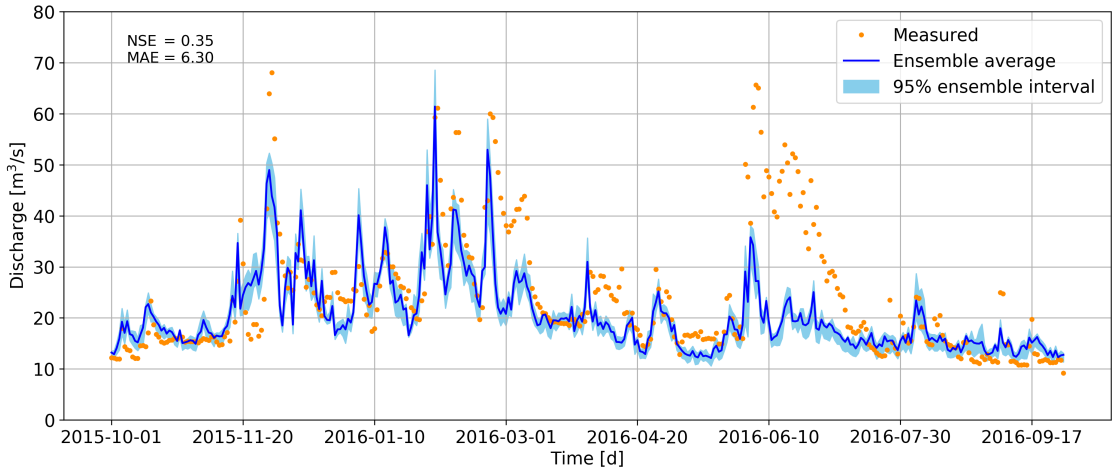

Figure B.6: Hydrograph of measured discharge and 2-days-ahead forecast of daily maximum discharge using gradient boosting.

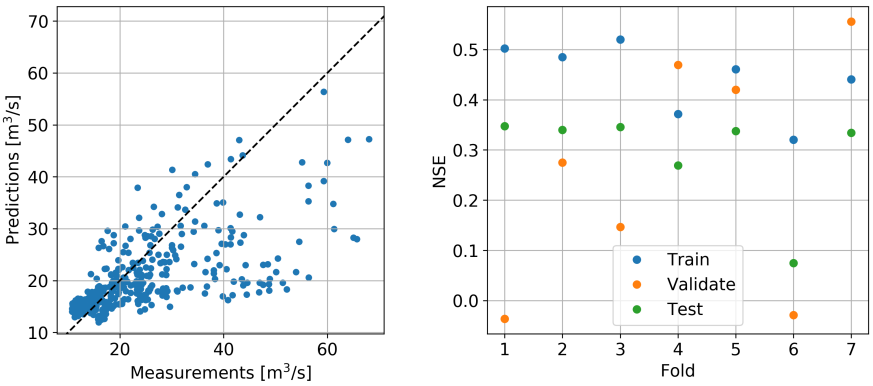

Figure B.7: 3-days-ahead forecast of daily maximum discharge using gradient boosting. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

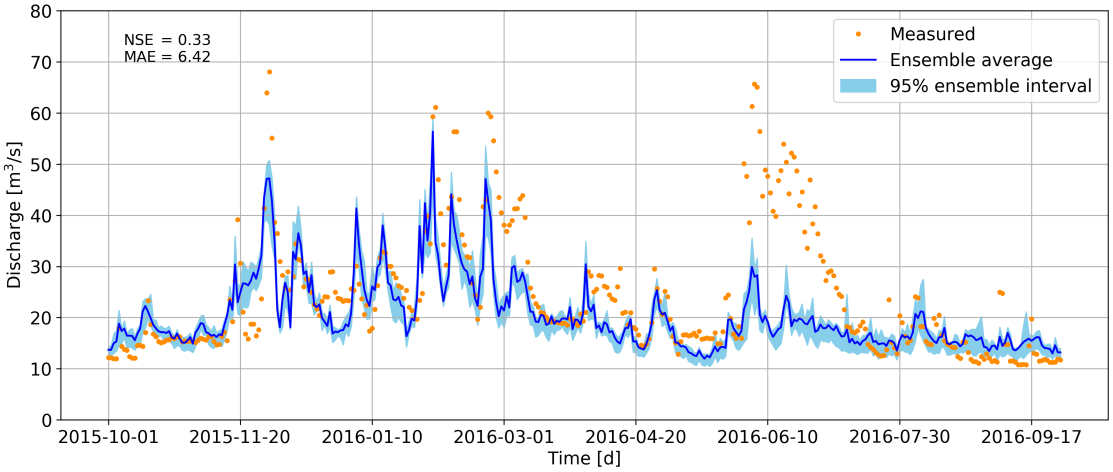

Figure B.8: Hydrograph of measured discharge and 3-days-ahead forecast of daily maximum discharge using gradient boosting.

B.3. DEEP LEARNING: GEUL

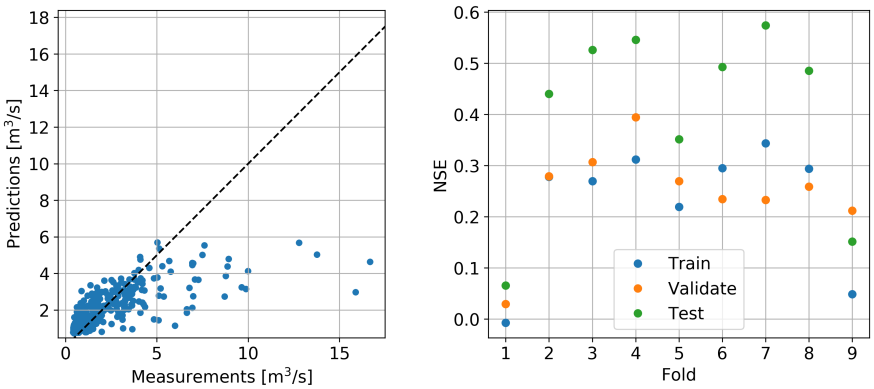

Figure B.9: 2-days-ahead forecast of daily maximum discharge using deep learning. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

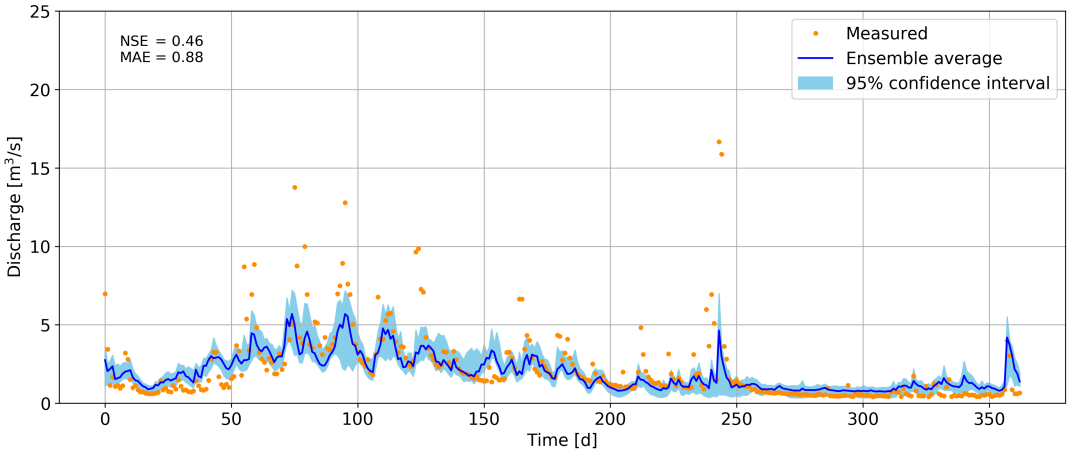

Figure B.10: Hydrograph of measured discharge and 2-days-ahead forecast of daily maximum discharge using deep learning.

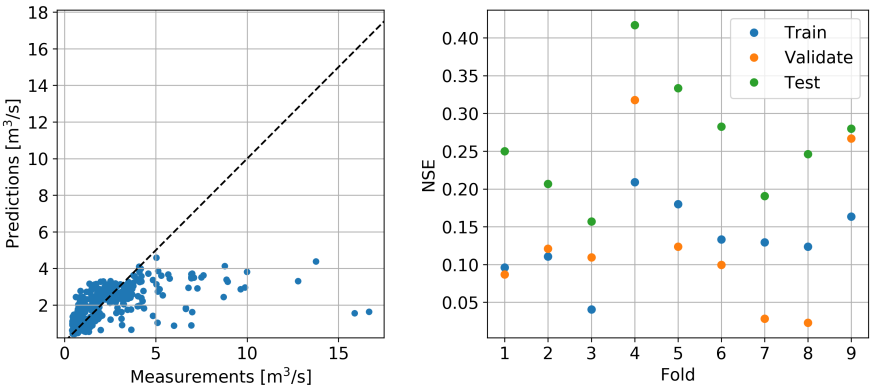

Figure B.11: 3-days-ahead forecast of daily maximum discharge using deep learning. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

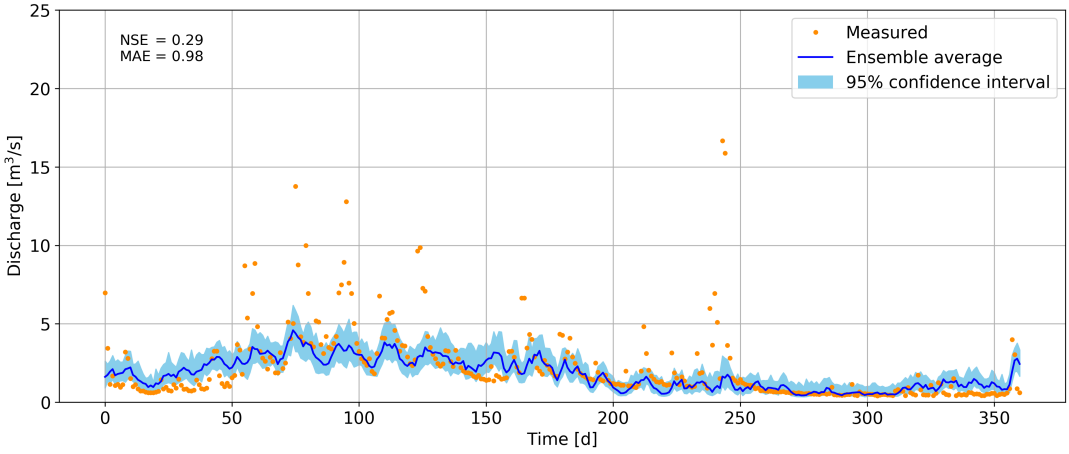

Figure B.12: Hydrograph of measured discharge and 3-days-ahead forecast of daily maximum discharge using deep learning.

B.4. DEEP LEARNING: RUR

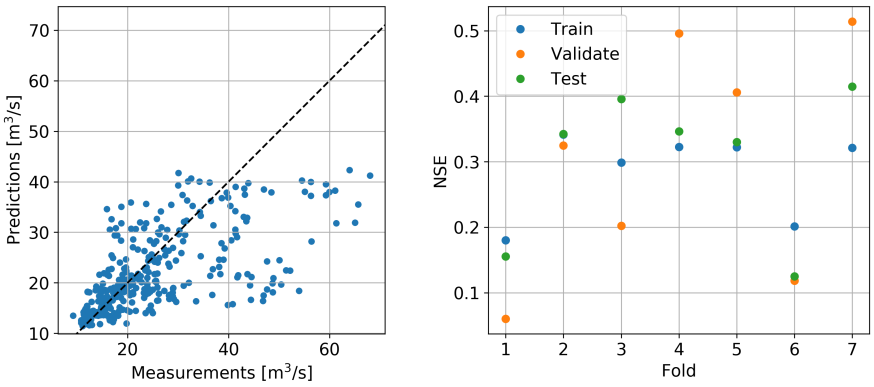

Figure B.13: 2-days-ahead forecast of daily maximum discharge using deep learning. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

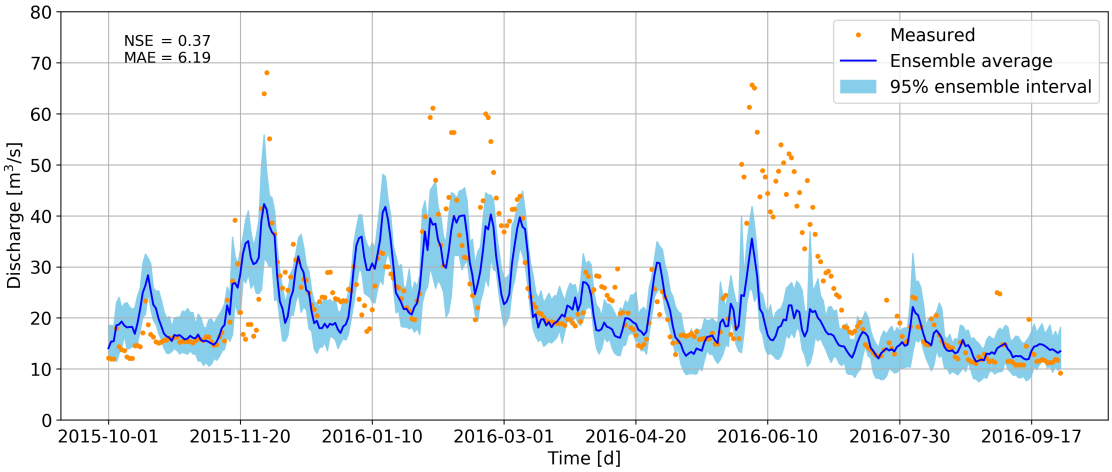

Figure B.14: Hydrograph of measured discharge and 2-days-ahead forecast of daily maximum discharge using deep learning.

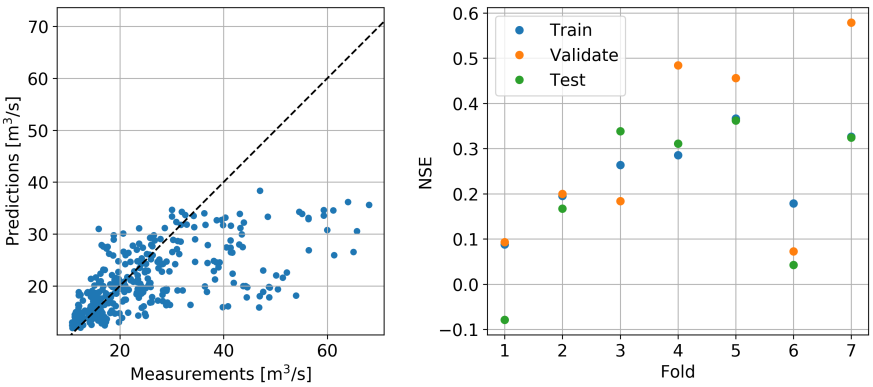

Figure B.15: 3-days-ahead forecast of daily maximum discharge using deep learning. Left: scatterplot of discharge measurements and forecast. Right: NSE for each fold.

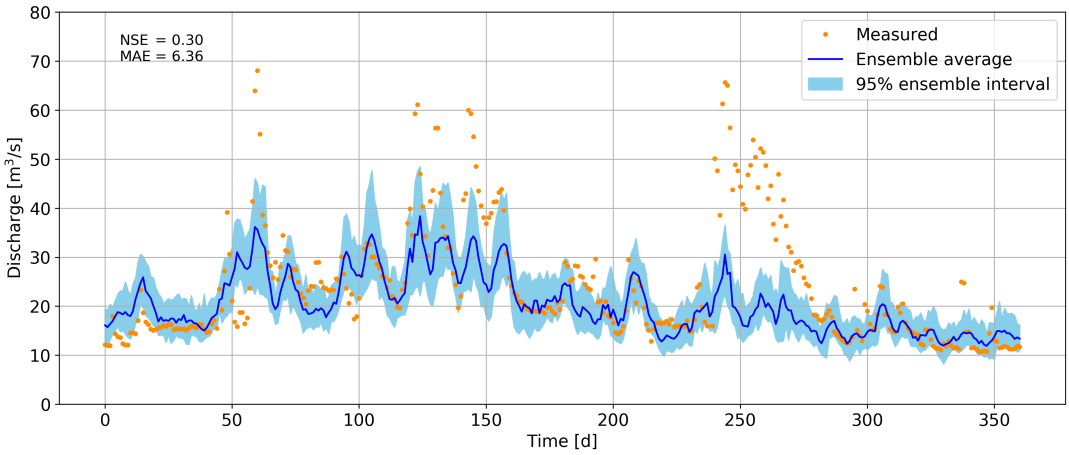

Figure B.16: Hydrograph of measured discharge and 3-days-ahead forecast of daily maximum discharge using deep learning.

B.5. RESULTS WATER DEPTH

In this section the results of modelling water depth are given.

B.5.1. GRADIENT BOOSTING - RUR

The performance of simulating daily maximum water depth (Figure B.18) is slightly worse compared to simulating daily maximum discharge (Figure 4.17). Again, peak timing is accurate but the magnitude is underestimated. The same period (2016-06-10 to 2016-07-20) performs the worst of the hydrograph. Whether this underestimation is due to the rating curve or a systematic underestimation is hard to tell.

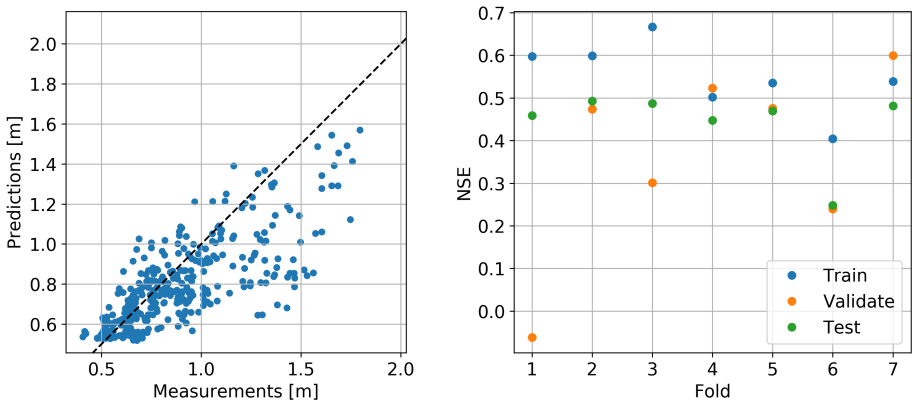

Figure B.17: Simulated maximum water depth using XGBoost. Left: scatterplot of water depth measurements and predictions. Right: NSE for each fold.

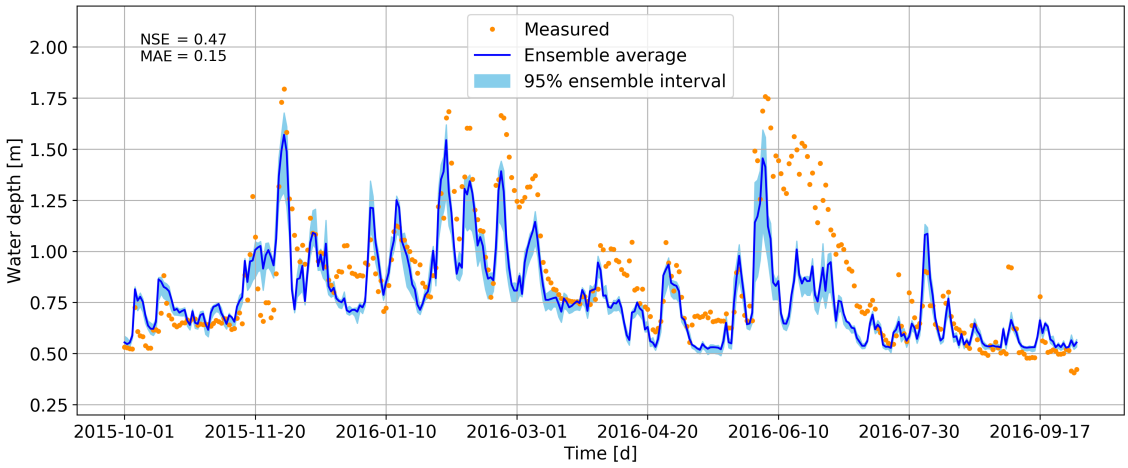

Figure B.18: Hydrograph of measured water depth and simulated maximum water depth of today using XGBoost.

B.5.2. DEEP LEARNING - GEUL

Water depth is also modelled instead of discharge for the same reason as given above; a smaller prediction range. Solely by looking at the scatter plot in Figure 4.8 one can notice that most points are under the dashed line, indicating underestimation of the water depth. This is confirmed by looking at the hydrograph at Figure 4.9. Especially low flows and falling limbs are underestimated. The model's performance is sufficient with an NSE of 0.53 and the mean absolute error is 0.07 centimetres. The peak timing is on point and peak magnitude is also acceptable. Actually, the only difference with predicting discharge in Figure 4.3 is that the output variable is generally underestimated here, where it is overestimated at discharge. The likely explanation is simply the rating curve, as given in Section 4.2.1.

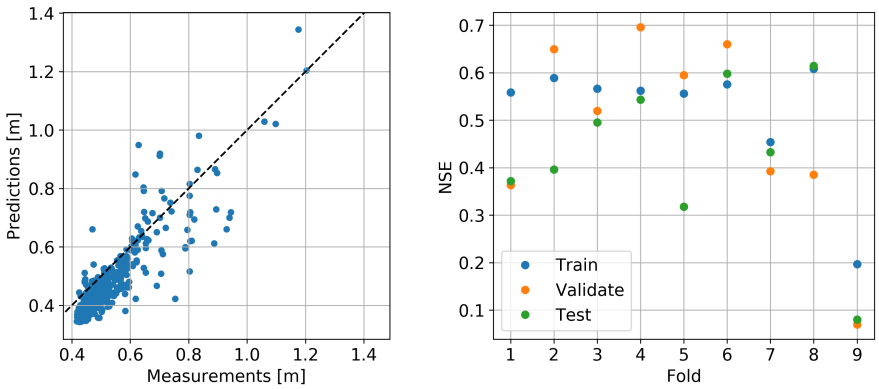

Figure B.19: simulated maximum water depth using LSTM. Left: scatterplot of water depth measurements and predictions. Right: NSE for each fold.

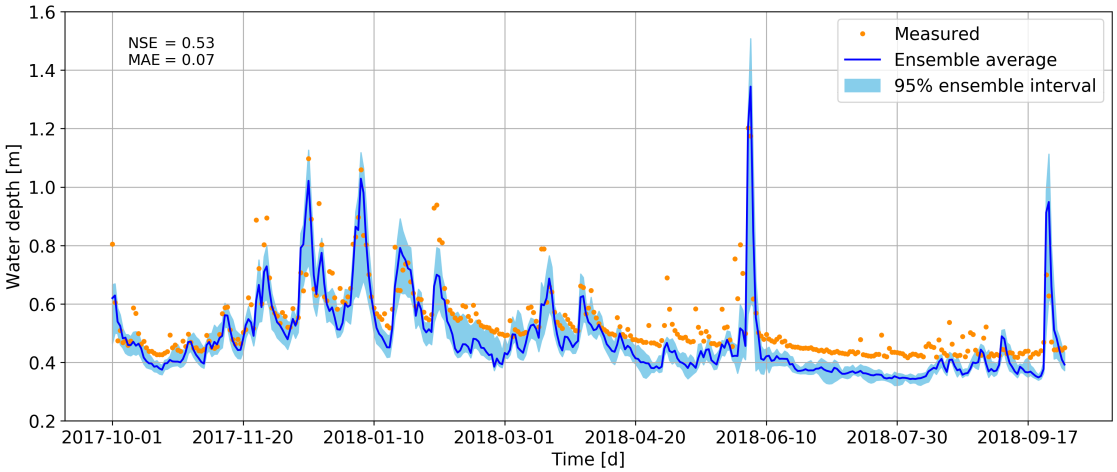

Figure B.20: Hydrograph of measured water depth and simulated maximum water depth using LSTM.

B.5.3. DEEP LEARNING - RUR

Water depth follows former results in that daily maximum water depth is underestimated too. The NSE of 0.45 is lower compared to the 0.51 of the daily maximum discharge. With an NSE below 0.50, the model's performance is classified as insufficient.

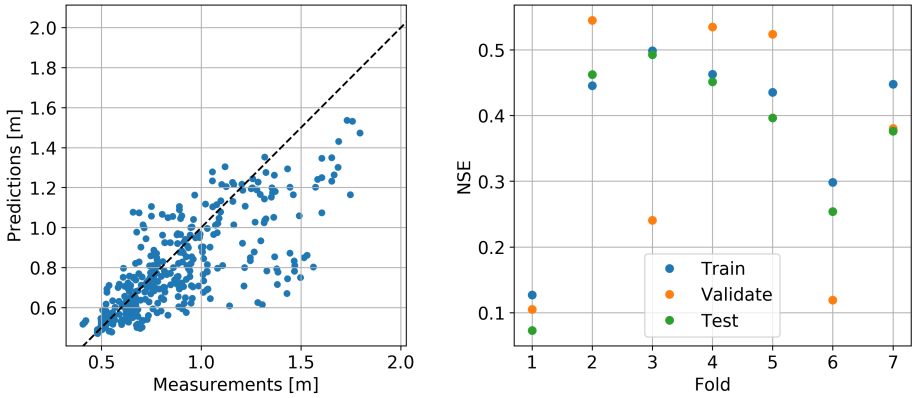

Figure B.21: simulated maximum water depth using LSTM. Left: scatterplot of water depth measurements and predictions. Right: NSE for each fold.

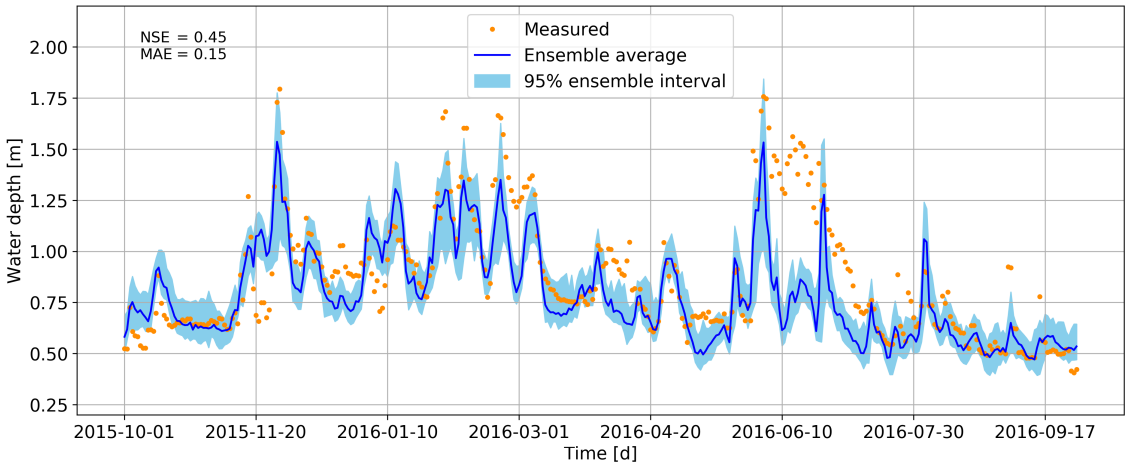

Figure B.22: Hydrograph of measured water depth and simulated maximum water depth using LSTM.

C

APPENDIX CODE

Listing C.1: GR4J conceptual model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from gr4j import gr4j
from scipy.optimize import differential_evolution
from scipy.stats import norm
from time import time

plt.rcParams.update({'font.size': 14})

def RunModel(rainfall, potential_evap, params):
        '''
        Parameter 1: storage capacity first reservoir (production reservoir) [mm]
        Parameter 2: catchment water exchange coefficient [mm/day]
        Parameter 3: one-day maximal capacity of routing reservoir [mm]
        Parameter 4: HU1 unit hydrograph time base [days]
        '''
        rainfall = rainfall
        potential_evap = potential_evap
        params = {'X1':params[0],
        'X2':params[1],
        'X3':params[2],
        'X4':params[3],
        }
        sim_discharge = gr4j(rainfall, potential_evap, params)
        return np.array(sim_discharge);

def MSE(sim,obs):
        '''
        sim: simulated
        obs: observed
        '''
        return np.mean(np.square(sim-obs));

def NSE(sim,obs):
        '''
        sim: simulated
        obs: observed
        '''
        return (1 - sum((sim-obs)**2)/sum((obs-np.mean(obs))**2));


def ObjectiveFunctionMSE(InputParams):
        '''
        InputData should be an array of parameter values
        '''
        obs = Q_t
        evap = evap_t
```

```
        rain = rain_t
        sim = RunModel(rain, evap, InputParams)
        return MSE(sim[365:],obs[365:])

## Load data, convert to mm/day
Geul = pd.read_csv('Discharge_Hommerich_1h_MD_OC.csv', delimiter=',', index_col=0, parse_dates=True,
skiprows=0, usecols=[0,1])
knmi = pd.read_csv('KNMI_processed.csv', delimiter=',', usecols=['ET'])
Geul[knmi.columns[0]] = knmi.values[:,0]
Geul['rain'] = np.loadtxt(r'Rain_Geul.csv')
dfx = Geul.resample('D').agg({'Discharge':max, 'ET':np.sum, 'rain':np.sum})
dfx.drop(pd.Timestamp('2018-04-30 00:00:00'), inplace=True)

n_forecast = 0
if n_forecast > 0:
dfx['Discharge'] = pd.Series(dfx['Discharge']).shift(-n_forecast)
dfx.drop(dfx.tail(n_forecast).index, inplace=True)
data = dfx.to_numpy()

## Convert max hourly discharge to mm/day: multiply by 3600 * 24 * 1000 / area (152 km2)
cmsmmd = 60 * 60 * 1000 * 24 / (152 * 1000 * 1000)
mmdcms = 1 / cmsmmd
data[:,0] = data[:,0] * cmsmmd

train = data[:int(np.shape(data)[0]*0.9),:]
val = data[int(np.shape(data)[0]*0.9)+1:,:]

Q_t = train[:,0]
evap_t = train[:,1]
rain_t = train[:,2]

Q_v = val[:,0]
evap_v = val[:,1]
rain_v = val[:,2]

## Run the model
start = time()
bounds = [(100,1200), (-5,3), (0,300), (1.1, 2.9)]
result = differential_evolution(ObjectiveFunctionNSE, bounds)
print('The calibrated values are: \nX1 = {} \nX2 = {} \nX3 = {} \nX4 = {}\nand\nNSE = {}'.format(round(result.x[0],0),
round(result.x[1],1), round(result.x[2],0), round(result.x[3],2), round(-result.fun,2)))

## Plot the results
sim_t = RunModel(rain_t, evap_t, result.x)
sim_v = RunModel(data[:,2], data[:,1], result.x)
sim_v = sim_v[int(np.shape(data)[0]*0.9)+1:]

stringo_t = 'NSE = {0:1.2f}'.format(NSE(sim_t[365:] * mmdcms, Q_t[365:] * mmdcms))
stringo_v = 'NSE = {0:1.2f}'.format(NSE(sim_v * mmdcms, Q_v * mmdcms))

stringo1 = 'NSE\nMAE'
stringo2 = '= {0:1.2f}\n= {1:1.2f}'.format(NSE(sim_v * mmdcms, Q_v * mmdcms), MAE(sim_v * mmdcms, Q_v * mmdcms))

fig = plt.figure(figsize=(15,6))
ax = fig.add_subplot(111)
plt.plot(data[:,0] * mmdcms, 'o', markersize='3', color='darkorange', label='Measured')
plt.plot(sim_t * mmdcms, label='Simulations - calibration')
plt.plot(list(range(int(np.shape(data)[0]*0.9), int(np.shape(data)[0]*0.9)+np.shape(sim_v)[0])),
sim_v * mmdcms, color='blue', label='Simulations - test')

plt.axvline(x=365, color='r', linestyle='--')
plt.axvline(x=int(np.shape(data)[0]*0.9), color='r', linestyle='--')
plt.grid()
plt.text(0.017, 0.94, 'Warm up', transform = ax.transAxes, fontsize=12)
plt.text(0.15, 0.94, 'Calibration', transform = ax.transAxes, fontsize=12)
plt.text(0.876, 0.94, 'Test', transform = ax.transAxes, fontsize=12)
plt.text(0.15, 0.89, stringo_t, transform = ax.transAxes, fontsize=12)
plt.text(0.876, 0.89, stringo_v, transform = ax.transAxes, fontsize=12)
plt.xlabel('Time [d]')
plt.ylabel('Discharge [m$^3$/s]')
plt.legend(loc=10, bbox_to_anchor=(0.72,0.88))
```

```
plt.savefig(path_fig + name +'_tvt.png', dpi=300, bbox_inches='tight');

fig2 = plt.figure(figsize=(15,6))
ax = fig2.add_subplot(111)
plt.grid()
plt.plot(data[int(np.shape(data)[0]*0.9)+1:,0] * mmdcms, 'o', markersize='3', color='darkorange', label='Measured')
plt.plot(sim_v * mmdcms, color='blue', label='Simulation')
plt.ylim((0,25))
ax.text(0.06, 0.87, stringo1, transform = ax.transAxes, fontsize=12)
ax.text(0.095, 0.87, stringo2, transform = ax.transAxes, fontsize=12)
#plt.title('GR4J model with today\'s discharge, test set only')
plt.xlabel('Time [d]')
plt.ylabel('Discharge [m$^3$/s]')
plt.savefig(path_fig + name +'_test.png', dpi=300, bbox_inches='tight')
plt.legend();

plt.figure(figsize=(12.5,5));
plt.grid()
plt.scatter(data[int(np.shape(data)[0]*0.9)+1:,0] * mmdcms, sim_v * mmdcms, s=20)
plt.xlabel('Measurements [m$^3$/s]')
plt.ylabel('Predictions [m$^3$/s]')
plt.axis('equal')
plt.axis('square')
plt.plot([-100, 100], [-100, 100], color='black', linestyle='--')
plt.ylim((0,20))
plt.xlim((0,20))
plt.savefig(path_fig + name +'_sp.png', dpi=300, bbox_inches='tight');
print('Computation took {0:1.0f} seconds.'.format(time() - start))
```

Listing C.2: ERA5 precipitation

```
# Load modules
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import os
import netCDF4
import xarray as xr

## Coordinates (lat/lon) of partial Rur catchment. Calculate weights for each raster cell. Area comes from QGIS shapefile.
# Divide each area by total to get weight.
lonG = [6]
latG = [50.75]

lonR = [6, 6.25, 6.5, 6, 6.25]
latR = [51, 51, 51, 50.75, 50.75]
AreaRur = np.array([75038674, 374292779, 35831151, 109512417, 94306855])
weightsRur = AreaRur / AreaRur.sum()

## Read in the netCDF4 file using xarray. Then select all values at certain coordinates, convert from m to mm,
# and add them to an array. The rows are hours, columns are locations. First location is upperleft, second is
# middle left etc. last one is bottomright.
files = os.listdir(path)
plt.figure(figsize=(15,10))

for file_nr in range(len(files)):
rain = xr.open_dataset(files[file_nr])
PrecipValuesRur = np.empty([np.shape(rain['time'].values)[0],len(lonR)])

for i in range(len(latR)):
rainlocRur = rain.sel(longitude=lonR[i], latitude=latR[i], method='nearest')*1000
PrecipValuesRur[:,i] = rainlocRur['tp'].data

RurRain = []
for row in range(len(PrecipValuesRur)):
RurRain.append(np.dot(weightsRur, PrecipValuesRur[row]))

rainlocGeul = rain.sel(longitude=lonG[0], latitude=latG[0], method='nearest')*1000
GeulRain = rainlocGeul['tp'].data
```

```
plt.plot(GeulRain, label=2008+file_nr);
print('{}: There are {} hours, of which {} non-zero. During {} hours it did not rain, which is {:1.0f}% of the total.'
.format(files[file_nr][7:11], len(GeulRain), np.count_nonzero(GeulRain), len(GeulRain)-np.count_nonzero(GeulRain), 100*(
filenameGeul = 'Rain_' + str(files[file_nr][7:11] + '_Geul.csv')

plt.title('Hourly rainfall over time for five locations.')
plt.xlabel('Time')
plt.ylabel('Total hourly precipitation [mm]')
plt.legend();

RainRur = []
RainGeul = []

for i in range(11):
fileRur = 'Rain_' + str(i+2008) + '_Rur.csv'
fileGeul = 'Rain_' + str(i+2008) + '_Geul.csv'
arrRur = np.loadtxt(fileRur)
arrGeul = np.loadtxt(fileGeul)
RainRur = np.concatenate([RainRur, arrRur])
RainGeul = np.concatenate([RainGeul, arrGeul])

np.savetxt('Rain_Rur.csv', RainRur, delimiter=',')
np.savetxt('Rain_Geul.csv', RainGeul, delimiter=',')
```

Listing C.3: Gradient boosting

```
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
import xgboost as xgb
import sklearn as skl
from sklearn import preprocessing
from datetime import datetime
from time import time
from bayes_opt import BayesianOptimization

def LaggedFeaturesBuilder(series, lag, dropna=False):
        result = pd.concat([series.shift(i) for i in np.arange(1, lag+1, 1)], axis=1)
        result.columns = ['{}_lag{}'.format(series.name, i) for i in np.arange(1, lag+1, 1)]
        if dropna:
        return result.dropna()
        else:
        return result;

def NSE(s,o):
        """
        Nash Sutcliffe efficiency coefficient
        input:
        s: simulated
        o: observed
        output:
        ns: Nash Sutcliffe efficient coefficient
        """
        return (1 - sum((s-o)**2)/sum((o-np.mean(o))**2))

def MAE(sim,obs):
        '''
        sim: simulated
        obs: observed
        '''
        return np.mean(abs(sim-obs));

def FitPredictPlot(dfx, dfy, regressor, n_folds, n_forecast, confperc, name):
        '''
        Fits the models to the dataset and predicts using the test set.
        '''
        start = time()
```

```
path_fig =

x_train = dfx.loc[:'2017-09-30']
y_train = dfy.loc[:'2017-09-30']
x_test = dfx.loc['2017-10-01':]
y_test_df = dfy.loc['2017-10-01':]

if n_forecast == 0:
y_test = y_test_df.values.flatten()
else:
y_test = y_test_df.values.flatten()[:-n_forecast]
y_test_df.drop(y_test_df.tail(n_forecast).index, inplace=True)

n_val = int(x_train.shape[0] / n_folds)

dfX = pd.DataFrame(index=['train', 'val'], columns=[str(i) for i in list(range(n_folds))])
dfY = pd.DataFrame(index=['train', 'val'], columns=[str(i) for i in list(range(n_folds))])

for i in range(n_folds):
x_v = x_train[i*n_val:(i+1)*n_val]
x_t = x_train.drop(index=x_v.index, axis=0)
dfX.iat[0,i] = x_t
dfX.iat[1,i] = x_v

y_v = y_train.loc[x_v.index]
y_t = y_train.drop(index=x_v.index, axis=0)
dfY.iat[0,i] = y_t
dfY.iat[1,i] = y_v

regressor = regressor
nsetvt = np.zeros((dfX.shape[1], 3))
maetvt = np.zeros((dfX.shape[1], 3))

if n_forecast == 0:
ensemble = np.zeros((len(y_test), dfX.shape[1]))
for i in range(dfX.shape[1]):
regressor.fit(dfX.iloc[0,i], dfY.iloc[0,i], eval_set=[(dfX.iloc[1,i], dfY.iloc[1,i])],
verbose=0, early_stopping_rounds=10)
predic = regressor.predict(x_test, ntree_limit=regressor.best_ntree_limit)
ensemble[:,i] = predic
#plt.plot(predic, label=i)
nsetvt[i,0] = NSE(regressor.predict(dfX.iloc[0,i]), dfY.iloc[0,i].values.flatten())
nsetvt[i,1] = NSE(regressor.predict(dfX.iloc[1,i]), dfY.iloc[1,i].values.flatten())
nsetvt[i,2] = NSE(predic, y_test)
maetvt[i,0] = MAE(regressor.predict(dfX.iloc[0,i]), dfY.iloc[0,i].values.flatten())
maetvt[i,1] = MAE(regressor.predict(dfX.iloc[1,i]), dfY.iloc[1,i].values.flatten())
maetvt[i,2] = MAE(predic, y_test)
else:
ensemble = np.zeros((len(y_test), dfX.shape[1]))
for i in range(dfX.shape[1]):
regressor.fit(dfX.iloc[0,i], dfY.iloc[0,i], eval_set=[(dfX.iloc[1,i], dfY.iloc[1,i])],
verbose=0, early_stopping_rounds=10)
predic = regressor.predict(x_test, ntree_limit=regressor.best_ntree_limit)[n_forecast:]
ensemble[:,i] = predic
#plt.plot(predic, label=i)
nsetvt[i,0] = NSE(regressor.predict(dfX.iloc[0,i])[n_forecast:], dfY.iloc[0,i].values.flatten()[:-n_forecast])
nsetvt[i,1] = NSE(regressor.predict(dfX.iloc[1,i])[n_forecast:], dfY.iloc[1,i].values.flatten()[:-n_forecast])
nsetvt[i,2] = NSE(predic, y_test)
maetvt[i,0] = MAE(regressor.predict(dfX.iloc[0,i])[n_forecast:], dfY.iloc[0,i].values.flatten()[:-n_forecast])
maetvt[i,1] = MAE(regressor.predict(dfX.iloc[1,i])[n_forecast:], dfY.iloc[1,i].values.flatten()[:-n_forecast])
maetvt[i,2] = MAE(predic, y_test)
print('NSE train / val / test: ' + str(np.round(nsetvt.mean(axis=0),2)))
print('MAE train / val / test: ' + str(np.round(maetvt.mean(axis=0),2)))

mu = np.quantile(ensemble, 0.5, axis=1)
y_test_df['Prediction'] = mu
y_test_df['hi'] = np.quantile(ensemble, 0.5-(confperc/200), axis=1)
y_test_df['lo'] = np.quantile(ensemble, 0.5+(confperc/200), axis=1)

stringo1 = 'NSE\nMAE'
stringo2 = '= {0:1.2f}\n= {1:1.2f}'.format(NSE(mu, y_test), MAE(mu, y_test))
```

```
        fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12.5,5))
        ax1.grid()
        ax1.scatter(y_test, mu, s=20)
        ax1.set_xlabel('Measurements [m$^3$/s]')
        ax1.set_ylabel('Predictions [m$^3$/s]')
        ax1.axis('equal')
        ax1.axis('square')
        ax1.plot([-100, 100], [-100, 100], color='black', linestyle='--');

        ax2.grid()
        ax2.plot(range(1,n_folds+1), nsetvt[:,0], 'o', label='Train')
        ax2.plot(range(1,n_folds+1), nsetvt[:,1], 'o', label='Validate')
        ax2.plot(range(1,n_folds+1), nsetvt[:,2], 'o', label='Test')
        ax2.set_xlabel('Fold')
        ax2.set_ylabel('NSE')
        ax2.set_xticks(np.arange(1, n_folds+1))
        ax2.legend();
        plt.savefig(path_fig + name +'_1.png', dpi=300, bbox_inches='tight')

        fig = plt.figure(figsize=(15,6))
        ax = fig.add_subplot(111)
        plt.grid()
        ax.fill_between(y_test_df.index, y_test_df.lo, y_test_df.hi, color='skyblue',
        label=str(confperc)+'% ensemble interval')
        ax.plot(y_test_df['Discharge'],'o', markersize='3', color='darkorange', label='Measured')
        ax.plot(y_test_df['Prediction'], color='blue', label='Ensemble Median')
        plt.xlabel('Time [d]')
        plt.ylabel('Discharge [m$^3$/s]')
        plt.ylim((0,25))
        ax.text(0.06, 0.87, stringo1, transform = ax.transAxes, fontsize=12)
        ax.text(0.095, 0.87, stringo2, transform = ax.transAxes, fontsize=12)
        plt.legend();
        plt.savefig(path_fig + name +'_2.png', dpi=300, bbox_inches='tight')
        print('Computation took {0:1.0f} seconds.'.format(time() - start))
        return y_test_df.Discharge, y_test_df.Prediction, y_test_df.hi, y_test_df.lo;

        plt.rcParams.update({'font.size': 14})

## Preprocessing and loading data
Geul = pd.read_csv('Discharge_Hommerich_1h_MD_OC.csv', delimiter=',', index_col=0, parse_dates=True,
skiprows=0, usecols=[0,1])
knmi = pd.read_csv('KNMI_processed.csv', delimiter=',', usecols=['T', 'FX', 'P', 'U', 'ET'])

for i in range(knmi.shape[1]):
Geul[knmi.columns[i]] = knmi.values[:,i]

Geul['rain'] = np.loadtxt(r'Rain_Geul.csv')
Geul['smc'] = np.repeat(np.loadtxt(r'SMC_Geul_MDNA.csv'), 24)
Geul['ndvi'] = np.repeat(np.loadtxt(r'NDVI_Geul.csv'),24)
Geul.replace(-9999, np.NaN, inplace=True)

dfx = Geul.resample('D').agg({'Discharge':max, 'T':np.mean, 'FX':np.mean, 'P':np.mean,
'U':np.mean, 'ET':np.sum, 'rain':np.sum, 'smc':np.mean, 'ndvi':np.mean})

## For hourly
#dfx.Geul.copy()

rwRain = 5
dfx['RainMA'] = dfx['rain'].rolling(window=rwRain).sum()
dfx['RainMA'][0:rwRain-1] = dfx['rain'][0:rwRain-1]
df_ET = LaggedFeaturesBuilder(pd.Series(dfx['ET']), 2, dropna=False)

dfx.drop(['P', 'U', 'FX', 'ndvi'], axis=1, inplace=True)
dfw = pd.concat([dfx, df_ET], axis=1)

dfw.drop(pd.Timestamp('2018-04-30 00:00:00'), inplace=True)
## For hourly
#dfx.drop(dfx.loc['2018-04-30 01:00:00': '2018-05-01 20:00:00'].index,inplace=True)

dfw.dropna(subset=['Discharge'], inplace=True)
```

```
dfy = dfw['Discharge'].copy().to_frame()
#dfy['Discharge'] = np.log(dfy['Discharge'])
dfw.drop(['Discharge'], axis=1, inplace=True)

n_forecast = 3
n_folds = 9

df_y = pd.Series(dfy['Discharge']).shift(-n_forecast).to_frame()
df_y.drop(df_y.tail(n_forecast).index, inplace=True)
dfw.drop(dfw.tail(n_forecast).index, inplace=True)

dfw.head()

## Actual training and predicting
regressor = xgb.XGBRegressor(objective='reg:squarederror',
colsample_bytree=0.898, learning_rate=0.0139, max_depth=6, n_estimators=1176, subsample=0.923, gamma=0)
y_test, mu, hi, lo = FitPredictPlot(dfw, df_y, regressor, n_folds, n_forecast, confperc=95, name='Geul_Q_d_0')

# Random search
x_train, x_val, x_test = tvtsplit2(dfw, 70, 20, 10)
y_train, y_val, y_test = tvtsplit2(df_y, 70, 20, 10)

pbounds = {
'learning_rate': (0.01, 0.03),
'n_estimators': (400, 1200),
'max_depth': (4, 8),
'subsample': (0.5, 1.0),
'colsample_bytree': (0.6, 1.0)}

def xgboost_hyper_param(learning_rate, n_estimators, max_depth, subsample, colsample_bytree):

        max_depth = int(max_depth)
        n_estimators = int(n_estimators)

        tst = xgb.XGBRegressor(
        objective ='reg:squarederror',
        max_depth=max_depth,
        learning_rate=learning_rate,
        n_estimators=n_estimators,
        colsample_bytree=colsample_bytree,
        subsample=subsample)

        tst.fit(x_train, y_train, early_stopping_rounds=20, eval_metric="mean_square_error",
        eval_set=[(x_train, y_train), (x_val, y_val)], verbose=0)
        predic = tst.predict(x_val, ntree_limit=tst.best_ntree_limit)
        metric = NSE(predic, y_val.values.flatten())
        return metric

        optimizer = BayesianOptimization(f=xgboost_hyper_param, pbounds=pbounds, random_state=1)
        optimizer.maximize(init_points=50, n_iter=0)
        optimizer.max

# If log of discharge is used to convert back to normal discharge
mua = np.exp(mu)
y_testa = np.exp(y_test.values.flatten())

stringo1 = 'NSE\nMAE'
stringo2 = '= {0:1.2f}\n= {1:1.2f}'.format(NSE(mua, y_testa), MAE(mua, y_testa))

fig = plt.figure(figsize=(15,6))
ax = fig.add_subplot(111)
ax.fill_between(y_test.index, np.exp(lo), np.exp(hi), color='skyblue', label='95% ensemble interval')
ax.plot(np.exp(y_test),'o', markersize='3', color='darkorange', label='Measured')
ax.plot(np.exp(mu), color='blue', label='Ensemble average')
plt.xlabel('Time [d]')
plt.ylabel('Discharge [m$^3$/s]')
plt.ylim((0,25))

ax.text(0.06, 0.87, stringo1, transform = ax.transAxes, fontsize=12)
ax.text(0.095, 0.87, stringo2, transform = ax.transAxes, fontsize=12)
plt.grid()
```

```
plt.legend();
```

Listing C.4: Retrieve KNMI data

```
from knmy import knmy
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os

## import data from KNMI for station 380 (Maastricht). Drop first row. Drop irrelevant columns, change datatype
disclaimer, station, variables, knmi = knmy.get_knmi_data(type='hourly', stations=[380], start=2008100100,
 end=2018093023, variables=['ALL'], inseason=True, parse=True)
knmi.drop(knmi.head(1).index, inplace=True)
knmi.drop(columns=['STN', 'DD', 'T10', 'SQ', 'VV', 'IX', 'WW', 'N'], inplace=True)

## Convert the units.
knmi = knmi.apply(pd.to_numeric)
knmi.loc[:, 'FH'] = knmi['FH'] / 10. # Mean wind velocity [m/s]
knmi.loc[:, 'FF'] = knmi['FF'] / 10. # Mean wind velocity over last 10 minutes of the hour [m/s]
knmi.loc[:, 'FX'] = knmi['FX'] / 10. # Maximum wind gust [m/s]
knmi.loc[:, 'T'] = knmi['T'] / 10. # Temperature [degrees Celsius]
knmi.loc[:, 'TD'] = knmi['TD'] / 10. # Dew point temperature [degrees Celsius]
knmi.loc[:, 'Q'] = knmi['Q'] * 10**4 # Global radiation [J/m2]
knmi.loc[:, 'RH'] = knmi['RH'] / 10. # Hourly precipitation amount [mm]
knmi.loc[:, 'P'] = knmi['P'] / 100. # Air pressure at MSL [kPa]
knmi.loc[:, 'U'] = knmi['U'] / 100. # Relative atmospheric humidity [%]

## Convert negative values at RH to 0 (KNMI: -1 means rainfall less than 0.05 mm).
knmi.loc[knmi['RH'] == -0.1, 'RH'] = 0

## Calculate evaporation
knmi['hov'] = (2501 - 2.375 * knmi['T']) * 1000    # Heat of vaporization [J/kg] dependence on temperature [C]
knmi['gamma'] = 0.0646 + 0.00006 * knmi['T']       # Psychrometer constant [kPa / C] dependence on temperature [C]
knmi['es'] = 0.6107 * 10**(7.5 * knmi['T'] /(237.3 + knmi['T'])) # Saturated vapor pressure relative to water [kPa]
knmi['s'] = (7.5 * 237.3) / ((237.3 + knmi['T'])**2) * np.log(10) * knmi.es # slope of the vapor pressure
curve [kPa / C]
knmi['ET'] = (0.62 * knmi['s'] * knmi['Q']) / (knmi['hov'] * (knmi['s'] + knmi['gamma']))

## Save dataframe to .csv file
#knmi.to_csv('KNMI_processed.csv')
```

Listing C.5: Deep learning model. Indenting is not always correct due to lack of space.

```
from sklearn import preprocessing
from sklearn import model_selection
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.models import Sequential, Model
from keras.layers import LSTM, Dropout, Embedding, Dense, Flatten
from keras import optimizers
import keras.backend as K
from keras.models import load_model
from time import time
from keras.callbacks import EarlyStopping

def split_sequences_adj(df, n_steps):
        '''
        This function makes samples of the input given the window of each sample
        (e.g. 3 means up to and including 3 days ago).
        Discharge has to be the first column of the dataframe!
        '''
        sequences = df
        X, y = list(), list()
        for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
```

```
        # check if we are beyond the dataset
        if end_ix > len(sequences):
        break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, 1:], sequences[end_ix-1, 0]
        X.append(seq_x)
        y.append(seq_y)
        return np.array(X), np.array(y);

def NSE(s,o):
        return (1 - (sum((s-o)**2)/sum((o-np.mean(o))**2)));

def MAE(sim,obs):
        return np.mean(abs(sim-obs));

def LSTM_date(dfx, n_steps, confperc, model, epochs, name, n_forecast, n_runs):

        start = time()
        path_fig =

        y_test_df = pd.Series(dfx['Discharge'].loc['2017-10-01':]).to_frame()
        ## Hourly
        #y_test_df = pd.Series(dfx['Discharge'].loc['2017-10-01 00:00:00':]).to_frame()
        y_test_df.dropna(inplace=True)

        if n_forecast > 0:
        dfx['Discharge'] = dfx['Discharge'].shift(-n_forecast)
        dfx.drop(dfx.tail(n_forecast).index, inplace=True)
        y_test_df.drop(y_test_df.tail(n_forecast).index, inplace=True)

        scalery = preprocessing.MinMaxScaler()
        scalerx = preprocessing.MinMaxScaler()
        dfx_scaled = scalerx.fit_transform(dfx.iloc[:, 1:])
        dfx_scaled[np.isnan(dfx_scaled)] = -1
        dfx_scaled = np.concatenate((scalery.fit_transform(dfx['Discharge'].values.reshape(-1,1)), dfx_scaled), axis=1)

        # Make samples
        x_train, y_train = split_sequences_adj(dfx_scaled, n_steps)
        # Drop samples where the target discharge is NaN
        x_train = x_train[~np.isnan(y_train)]
        y_train = y_train[~np.isnan(y_train)]

        if n_forecast == 0:
        #SplitIndex = np.where(y_train==scalery.transform(dfx['Discharge'].loc['2017-10-01'].reshape(-1, 1)))[1][0]
        ## Water depth
        #SplitIndex = np.where(y_train==scalery.transform(dfx['Discharge'].loc['2017-10-01'].reshape(-1, 1)))[1][4]
        ## Hourly discharge
        SplitIndex = np.where(y_train==scalery.transform(dfx['Discharge'].loc['2017-10-01 00:00:00'].reshape(-1, 1))) [1][0]

        elif n_forecast == 1:
        SplitIndex = 3284
        #SplitIndex = np.where(y_train==scalery.transform(dfx['Discharge'].loc['2017-10-01'].reshape(-1, 1)))[1][4]

        train_x = x_train[:SplitIndex, :]
        x_test = x_train[SplitIndex:, :]
        train_y = y_train[:SplitIndex]
        test_y = y_train[SplitIndex:]
        n_val = int(train_x.shape[0] / 9)
        maetvt = np.empty([n_folds, 3])
        nsetvt = np.empty([n_folds, 3])
        ensemble = np.empty([np.shape(x_test)[0], (n_runs*n_folds)])

        es = EarlyStopping(monitor='loss', mode='min', patience=10, verbose=0, restore_best_weights=True)

        if n_forecast == 0:
                y_test = scalery.inverse_transform(test_y.reshape(-1, 1)).flatten()
                else:
                y_test = scalery.inverse_transform(test_y.reshape(-1, 1)).flatten()#[:-n_forecast]

        global mothermodel
        mothermodel = model
```

```
for i in range(n_folds):
modelv = mothermodel
session = tf.Session()
with session.as_default():
session.run(tf.global_variables_initializer())
with session.graph.as_default():
x_train = np.delete(train_x, np.s_[i*n_val:(i+1)*n_val], axis=0)
y_train = np.delete(train_y, np.s_[i*n_val:(i+1)*n_val])

x_val = train_x[i*n_val:(i+1)*n_val,:]
y_val = train_y[i*n_val:(i+1)*n_val]

hist = modelv.fit(x_train, y_train, epochs=epochs, batch_size=64,
                                validation_data=(x_val, y_val), verbose=0, callbacks=[es]);
if n_forecast == 0:
preds = scalery.inverse_transform(modelv.predict(x_test)).flatten()
nsetvt[i,0] = NSE(scalery.inverse_transform(modelv.predict(x_train)),
                                scalery.inverse_transform(y_train.reshape(-1,1)))
nsetvt[i,1] = NSE(scalery.inverse_transform(modelv.predict(x_val)),
                                scalery.inverse_transform(y_val.reshape(-1,1)))
nsetvt[i,2] = NSE(preds, y_test)
maetvt[i,0] = MAE(scalery.inverse_transform(modelv.predict(x_train)),
                                scalery.inverse_transform(y_train.reshape(-1,1)))
maetvt[i,1] = MAE(scalery.inverse_transform(modelv.predict(x_val)),
                                scalery.inverse_transform(y_val.reshape(-1,1)))
maetvt[i,2] = MAE(preds, y_test)
modelv.save()
path_to_model =

        for run in range(n_runs):
        if run == 0:
        K.set_learning_phase(1)
        modelv = load_model(path_to_model)

        pred = modelv.predict(x_test)
        ensemble[:,(n_runs*i)+run] = scalery.inverse_transform(pred).flatten()
else:
preds = scalery.inverse_transform(modelv.predict(x_test)).flatten()#[n_forecast:]
nsetvt[i,0] = NSE(scalery.inverse_transform(modelv.predict(x_train))[n_forecast:],
                                scalery.inverse_transform(y_train.reshape(-1,1))[:-n_forecast])
nsetvt[i,1] = NSE(scalery.inverse_transform(modelv.predict(x_val))[n_forecast:],
                                scalery.inverse_transform(y_val.reshape(-1,1))[:-n_forecast])
nsetvt[i,2] = NSE(preds, y_test)
maetvt[i,0] = MAE(scalery.inverse_transform(modelv.predict(x_train))[n_forecast:],
                                scalery.inverse_transform(y_train.reshape(-1,1))[:-n_forecast])
maetvt[i,1] = MAE(scalery.inverse_transform(modelv.predict(x_val))[n_forecast:],
                                scalery.inverse_transform(y_val.reshape(-1,1))[:-n_forecast])
maetvt[i,2] = MAE(preds, y_test)
modelv.save()
path_to_model =

for run in range(n_runs):
        if run == 0:
        K.set_learning_phase(1)
        modelv = load_model(path_to_model)

        pred = modelv.predict(x_test)
        ensemble[:,(n_runs*i)+run] = scalery.inverse_transform(pred).flatten()

mu = np.quantile(ensemble, 0.5, axis=1)
print(y_test_df.shape, np.shape(mu))
y_test_df['Prediction'] = mu
y_test_df['hi'] = np.quantile(ensemble, 0.5+(confperc/200), axis=1)
y_test_df['lo'] = np.quantile(ensemble, 0.5-(confperc/200), axis=1)

print('NSE train / val / test: ' + str(np.round(nsetvt.mean(axis=0),2)))
print('MAE train / val / test: ' + str(np.round(maetvt.mean(axis=0),2)))
stringo1 = 'NSE\nMAE'
stringo2 = '= {0:1.2f}\n= {1:1.2f}'.format(NSE(mu, y_test), MAE(mu, y_test))

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12.5,5))
```

```
        ax1.grid()
        ax1.scatter(y_test, mu, s=20)
        ax1.set_xlabel('Measurements [m$^3$/s]') #[m$^3$/s]
        ax1.set_ylabel('Predictions [m$^3$/s]')
        ax1.axis('equal')
        ax1.axis('square')
        ax1.plot([-100, 100], [-100, 100], color='black', linestyle='--');

        ax2.grid()
        ax2.plot(range(1,n_folds+1), nsetvt[:,0], 'o', label='Train')
        ax2.plot(range(1,n_folds+1), nsetvt[:,1], 'o', label='Validate')
        ax2.plot(range(1,n_folds+1), nsetvt[:,2], 'o', label='Test')
        ax2.set_xlabel('Fold')
        ax2.set_ylabel('NSE')
        ax2.set_xticks(np.arange(1, n_folds+1))
        ax2.legend();
        plt.savefig(path_fig + name +'_1.png', dpi=300, bbox_inches='tight')

        fig = plt.figure(figsize=(15,6))
        ax = fig.add_subplot(111)
        plt.grid()
        ax.fill_between(y_test_df.index, y_test_df.lo, y_test_df.hi, color='skyblue',
         label=str(confperc)+'% ensemble interval')
        ax.plot(y_test_df['Discharge'],'o', markersize='3', color='darkorange', label='Measured')
        ax.plot(y_test_df['Prediction'], color='blue', label='Ensemble average')
        plt.xlabel('Time [h]')
        plt.ylabel('Discharge [m$^3$/s]')
        plt.ylim((0,25))
        ax.text(0.06, 0.87, stringo1, transform = ax.transAxes, fontsize=12)
        ax.text(0.095, 0.87, stringo2, transform = ax.transAxes, fontsize=12)
        plt.legend();
        plt.savefig(path_fig + name +'_2.png', dpi=300, bbox_inches='tight')
        print('Computational time took {0:1.2f} seconds.'.format(time()-start))
        return mu, y_test, y_test_df;

## Loading and preprocessing data
Geul = pd.read_csv('Discharge_Hommerich_1h_MD_OC.csv', delimiter=',', index_col=0, parse_dates=True, skiprows=0, usecols=[0,1])
## Water Depth
#Geul = pd.read_csv('Waterlevel_Hommerich_1h.csv', delimiter=',', index_col=0, parse_dates=True, skiprows=0, usecols=[0,1])
knmi = pd.read_csv('KNMI_processed.csv', delimiter=',', usecols=['T', 'FX', 'P', 'U', 'ET'])

for i in range(knmi.shape[1]):
Geul[knmi.columns[i]] = knmi.values[:,i]

Geul['rain'] = np.loadtxt(r'Rain_Geul.csv')
Geul['smc'] = np.repeat(np.loadtxt(r'SMC_Geul_MDNA.csv'),24)
Geul['ndvi'] = np.repeat(np.loadtxt(r'NDVI_Geul.csv'),24)
Geul.replace(-9999, np.NaN, inplace=True)

## Hourly
#dfx = Geul.copy()
dfx = Geul.resample('D').agg({'Discharge':np.max, 'T':np.mean, 'FX':np.mean, 'P':np.mean, 'U':np.mean, 'ET':np.sum,
 'rain':np.sum, 'smc':np.mean, 'ndvi':np.mean})

rwRain = 5
dfx['RainMA'] = dfx['rain'].rolling(window=rwRain).sum()
dfx['RainMA'][0:rwRain-1] = dfx['rain'][0:rwRain-1]

dfx.drop(columns=['U', 'P', 'FX', 'ndvi'], inplace=True)

## Logarithmic
#dfx['Discharge'] = np.log(dfx['Discharge'])
#dfx.drop(pd.Timestamp('2018-04-30 00:00:00'), inplace=True)
## Hourly
#dfx.drop(dfx.loc['2018-04-30 01:00:00': '2018-05-01 20:00:00'].index,inplace=True)
#dfx.drop(dfx.loc[:'2015-09-30 00:00:00'].index,inplace=True)

n_features = dfx.shape[1] - 1
n_folds = 9        # Kfold validation (approximately 9 years for training and validation, so fold 9 times)
n_steps = 4        # Steps for the LSTM to take into account, e.g. 3 means up to three days ago
n_runs = 5         # Amount of runs per fold for the confidence interval
```

```
n_forecast = 0     # How many time periods (in this case days) ahead we want to forecast
confinterval = 95 # Confidence interval.
epochs = 100       # Maximum amount of epochs per fold
dfx.head()

## Training and predicting
lstm_Geul = Sequential()
lstm_Geul.add(LSTM(64, activation='relu', return_sequences=True, input_shape=(n_steps, n_features)))
lstm_Geul.add(Dropout(rate=0.2))
lstm_Geul.add(LSTM(8, activation='relu', return_sequences=False))
lstm_Geul.add(Dense(1, activation='sigmoid'))
lstm_Geul.compile(loss='mean_squared_error', optimizer='adam', metrics=['mse'])
model = lstm_Geul

mu, y_test, y_test_df = LSTM_date(dfx, n_steps, confinterval, model, epochs, name=name,
 n_forecast=n_forecast, n_runs=n_runs)

## If logarithm of discharge is used
mua = np.exp(y_test_df.Prediction)
y_testa = np.exp(y_test_df.Discharge)

stringo1 = 'NSE\nMAE'
stringo2 = '= {0:1.2f}\n= {1:1.2f}'.format(NSE(mua, y_testa), MAE(mua, y_testa))

fig = plt.figure(figsize=(15,6))
ax = fig.add_subplot(111)
ax.fill_between(y_test_df.index, np.exp(y_test_df.lo), np.exp(y_test_df.hi), color='skyblue',
label='95% confidence interval')
ax.plot(mua, color='blue', label='Ensemble average')
ax.plot(y_testa,'o', markersize='3', color='darkorange', label='Measured')
plt.xlabel('Time [d]')
plt.ylabel('Discharge [m$^3$/s]')
ax.text(0.06, 0.87, stringo1, transform = ax.transAxes, fontsize=12)
ax.text(0.095, 0.87, stringo2, transform = ax.transAxes, fontsize=12)
plt.grid()
plt.ylim((0,25))
plt.legend();
path_fig = r'C:\Users\gijsv\1_Thesis\Images\Geul\LSTM_date-axis\\'

plt.savefig(path_fig + name +'_2_exp.png', dpi=300, bbox_inches='tight')
```

Listing C.6: ERA5 meteorological data processing

```
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
import xarray as xr
import netCDF4

variables = ['ENSW', 'NNSW', 'NSAT', 'DPT', 'SNSR', 'SLP']
varname = ['uas', 'vas', 'tas', 'dpt','ssr_NON_CDM', 'psl'] #ENSW, NNSW, NSAT, DPT, SNSR, SLP
periods = ['2008', '2009-12', '2013-17', '2018']

AreaRur = np.array([75038674, 374292779, 35831151, 109512417, 94306855])
weightsRur = AreaRur / AreaRur.sum()
meteo = pd.DataFrame()
for i in range(len(variables)):
total = []
for period in periods:
var = xr.open_dataset('{0}_{1}_mean.nc'.format(variables[i], period))
values = np.empty([np.shape(var['time'].values)[0], 5])
values[:,0] = var[(varname[i])][0,1][0].values
values[:,1] = var[(varname[i])][1,1][0].values
values[:,2] = var[(varname[i])][2,1][0].values
values[:,3] = var[(varname[i])][0,1][0].values
values[:,4] = var[(varname[i])][0,1][0].values
weighted = []
for row in range(len(values)):
weighted.append(np.dot(weightsRur, values[row]))
```

```
total.append(weighted)
flat_list = np.array([item for sublist in total for item in sublist])
meteo[variables[i]] = flat_list
#plt.figure()
#plt.plot(flat_list, label=variables[i])
#plt.legend()
#np.savetxt(str(variables[i])+'.txt', flat_list)

## Evaporation is very small (order of 0.6-1.5 mm/day)
meteo['FH'] = np.sqrt(meteo['ENSW']**2 + meteo['NNSW']** 2)
meteo['NSAT'] = meteo['NSAT'] - 273.15
meteo['DPT'] = meteo['DPT'] - 273.15
meteo['SLP'] = meteo['SLP'] / 1000
meteo.columns = ['ENSW', 'NNSW', 'T', 'DPT', 'Q', 'P', 'FH']

## Calculate evaporation
knmi['hov'] = (2501 - 2.375 * knmi['T']) * 1000    # Heat of vaporization [J/kg] dependence on temperature [C]
knmi['gamma'] = 0.0646 + 0.00006 * knmi['T']       # Psychrometer constant [kPa / C] dependence on temperature [C]
knmi['es'] = 0.6107 * 10**(7.5 * knmi['T'] /(237.3 + knmi['T'])) # Saturated vapor pressure relative to water [kPa]
knmi['s'] = (7.5 * 237.3) / ((237.3 + knmi['T'])**2) * np.log(10) * knmi.es # slope of the vapor pressure
curve [kPa / C]
knmi['ET'] = (0.62 * knmi['s'] * knmi['Q']) / (knmi['hov'] * (knmi['s'] + knmi['gamma']))

#meteo.drop(columns=['ENSW', 'NNSW', 'Q', 'hov', 'gamma', 'es', 's'], inplace=True)
#meteo.to_csv(r'C:\Users\gijsv\1_Thesis\Processed_data\ERA5_processed.csv')
```

Listing C.7: NDVI processing

```
import numpy as np
import matplotlib.pyplot as plt
import os
import netCDF4
import re
import wget
import xarray as xr
from datetime import datetime
import time

## The coordinates and arease for both catchments.
lonG = [5.93, 5.97, 5.88, 5.93, 5.97, 6.03, 5.88, 5.93, 5.97, 6.03, 6.08, 6.13, 5.93, 5.97, 6.03, 6.08, 6.13]
latG = [50.82, 50.82, 50.77, 50.77, 50.77, 50.77, 50.72, 50.72, 50.72, 50.72, 50.72, 50.72, 50.67, 50.67, 50.67, 50.67, 50.67]

lonR = [6.08, 6.13, 6.18, 6.23, 6.08, 6.13, 6.18, 6.23, 6.28, 6.33, 6.08, 6.13, 6.18, 6.23, 6.28, 6.33, 6.38, 6.43,
6.48, 6.08, 6.13, 6.18, 6.23, 6.28, 6.33, 6.38, 6.43, 6.48, 6.03, 6.08, 6.13, 6.18, 6.23, 6.28, 6.33, 6.38, 6.03, 6.08,
6.13, 6.18, 6.23, 6.28, 6.33, 6.03, 6.08, 6.13, 6.18, 6.23, 6.28, 6.33, 6.03, 6.08, 6.13, 6.18, 6.03, 6.08, 6.13, 6.18]
latR = [51.12, 51.12, 51.12, 51.12, 51.07, 51.07, 51.07, 51.07, 51.07, 51.07, 51.02, 51.02, 51.02, 51.02, 51.02,
51.02, 51.02, 51.02, 51.02, 50.97, 50.97, 50.97, 50.97, 50.97, 50.97, 50.97, 50.97, 50.97, 50.92, 50.92,
50.92, 50.92, 50.92, 50.92, 50.92, 50.92, 50.87, 50.87, 50.87, 50.87, 50.87, 50.87, 50.87, 50.82, 50.82,
50.82, 50.82, 50.82, 50.82, 50.82, 50.82, 50.77, 50.77, 50.77, 50.77, 50.72, 50.72, 50.72, 50.72]

AreaGeul = np.array([2198068, 29130, 4646636, 19419771, 7146676, 774581, 272419, 13689937,
19637676, 18679338, 13768753, 3848868, 4012223, 14396396, 16968222, 11909725, 1319080])
weightsGeul = AreaGeul / AreaGeul.sum()

AreaRur = np.array([75925, 2827223, 1562213, 407688, 674633, 18801724, 19448417,
16937771, 13276526, 3114741, 1131838, 16162981,
19513016, 19513016, 19513016, 18842781, 16157708, 13303158, 5412, 1861425, 19300739, 19533831, 19533831, 19533831, 19533831,
14283441, 7707661, 716406, 1272710, 11786884, 19554631, 19554631, 19554631, 19554631, 13315501, 3694655, 12707606, 19575415,
19575415, 19575415, 19575415, 15637405, 10478, 10133439, 19596185, 19030740, 13703778, 10959602, 5432186, 92271, 8388958,
19616938, 19480950, 9947577, 953963, 5868923, 6821558, 734600])
weightsRur = AreaRur / AreaRur.sum()

## Read in all the files, extract data to array.
start = time.time()
files = os.listdir(path)
NDVIvaluesGeul = np.empty([len(files), len(latG)])
NDVIvaluesRur = np.empty([len(files), len(latR)])
for file_nr in range(len(files)):
dataset = xr.open_dataset(files[file_nr])
for i in range(len(latG)):
```

```
dataset_loc = dataset.sel(longitude=lonG[i], latitude=latG[i], method='nearest')
NDVIvaluesGeul[file_nr,i] = dataset_loc['NDVI'].data
for i in range(len(latR)):
dataset_loc = dataset.sel(longitude=lonR[i], latitude=latR[i], method='nearest')
NDVIvaluesRur[file_nr,i] = dataset_loc['NDVI'].data
end = time.time()
print(end-start)

NDVIgeul = []
NDVIvaluesGeul[np.isnan(NDVIvaluesGeul)] = 0
for row in range(len(NDVIvaluesGeul)):
wi = 0
ndvi_unw = np.dot(weightsGeul, NDVIvaluesGeul[row])
for area in range(len(weightsGeul)):
if NDVIvaluesGeul[row,area] != 0:
wi += weightsGeul[area]
if wi != 0:
NDVIgeul.append(ndvi_unw / wi)
else:
NDVIgeul.append(0)
print('Geul: There are {} days, of which {} have a non-zero value. {} days are missing, which is {:1.2 f}% of
the total.'.format(len(NDVIgeul), np.count_nonzero(NDVIgeul), len(NDVIgeul)-np.count_nonzero(NDVIgeul),
100*(len(NDVIgeul)-np.count_nonzero(NDVIgeul))/len(NDVIgeul)))

NDVIrur = []
NDVIvaluesRur[np.isnan(NDVIvaluesRur)] = 0
for row in range(len(NDVIvaluesRur)):
wi = 0
ndvi_unw = np.dot(weightsRur, NDVIvaluesRur[row])
for area in range(len(latR)):
if NDVIvaluesRur[row,area] != 0:
wi += weightsRur[area]
if wi != 0:
NDVIrur.append(ndvi_unw / wi)
else:
NDVIrur.append(0)
print('Rur: There are {} days, of which {} have a non-zero value. {} days are missing, which is {:1.2 f}% of
the total.'.format(len(NDVIrur), np.count_nonzero(NDVIrur), len(NDVIrur)-np.count_nonzero(NDVIrur),
100*(len(NDVIrur)-np.count_nonzero(NDVIrur))/len(NDVIrur)))

filenameGeul = 'NDVI_' + str(dataset.time_coverage_start[:4]) + '_Geul.csv'
filenameRur = 'NDVI_' + str(dataset.time_coverage_start[:4]) + '_Rur.csv'

## Load all files for Geul and Rur and append to two seperate arrays.
NDVIgeul = []
NDVIrur = []

for i in range(11):
fileGeul = 'NDVI_' + str(i+2008) + '_Geul.csv'
fileRur = 'NDVI_' + str(i+2008) + '_Rur.csv'
arrGeul = np.loadtxt(fileGeul)
arrRur = np.loadtxt(fileRur)
NDVIgeul = np.concatenate([NDVIgeul, arrGeul])
NDVIrur = np.concatenate([NDVIrur, arrRur])

np.savetxt('NDVI_Geul.csv', NDVIgeul, delimiter=',')
np.savetxt('NDVI_Rur.csv', NDVIrur, delimiter=',')
```

Listing C.8: Soil moisture content processing

```
# Load modules
import numpy as np, import matplotlib.pyplot as plt, import pandas as pd, import os
import netCDF4, import xarray as xr

files = os.listdir(path)
SMCvaluesGeul = np.empty([len(files), 4])
SMCvaluesRur = np.empty([len(files), 5])
for i in range(len(files)):
dataset = netCDF4.Dataset(files[i], 'r')
SMCvaluesGeul[i,:] = dataset['sm'][0,156,743].data.item(), dataset['sm'][0,156,744].data.item(),
```

```
dataset['sm'][0,157,743].data.item(), dataset['sm'][0,157,744].data.item()
SMCvaluesRur[i,:] = dataset['sm'][0,155,744].data.item(), dataset['sm'][0,155,745].data.item(),
dataset['sm'][0,156,744].data.item(), dataset['sm'][0,156,745].data.item(),
dataset['sm'][0,157,744].data.item()

## Weights of each cell. 1 and 2 are upper row from left to right. 3 and 4 are lower row from left to right.
AreaGeul = np.array([33520314, 50917413, 846490, 67091877])
weightsGeul = AreaGeul / AreaGeul.sum()

# Not all cells are scanned each orbit. Therefore, one has to adjust for missing values.
# Create empty list to store daily averaged soil moisture content values in. Multiply weights with respective value
# using dot product as this is an efficient operation. Also, if area has not been scanned it will result in
# multiplying the weight by zero. Then check for each area if the value is non-zero. If this is the case, add weight
# of the area to 'wi'. If 'wi' is non-zero (so at least one area has a value), divide dot product by total weights.
# However, if no area has been scanned, set a zero.
# Finally, print the amount of non-zero values and compare this with the total possible amount of values
smcGeul = []
for row in range(len(SMCvaluesGeul)):
wi = 0
smc_unw = np.dot(weightsGeul, SMCvaluesGeul[row])
for area in range(4):
if SMCvaluesGeul[row,area] != 0:
wi += weightsGeul[area]
if wi != 0:
smcGeul.append(smc_unw / wi)
else:
smcGeul.append(0)
print('Geul: There are {} days, of which {} have a non-zero value. {} days are missing, which is {:1.2 f}% of
the total.'.format(len(smcGeul), np.count_nonzero(smcGeul), len(smcGeul)-np.count_nonzero(smcGeul),
100*(len(smcGeul)-np.count_nonzero(smcGeul))/len(smcGeul)))

smcRur = []
AreaRur = np.array([11695174, 83822410, 354217263, 119619931, 14006076])
weightsRur = AreaRur / AreaRur.sum()
for row in range(len(SMCvaluesRur)):
wi = 0
smc_unw = np.dot(weightsRur, SMCvaluesRur[row])
for area in range(5):
if SMCvaluesRur[row,area] != 0:
wi += weightsRur[area]
if wi != 0:
smcRur.append(smc_unw / wi)
else:
smcRur.append(0)
print('Rur: There are {} days, of which {} have a non-zero value. {} days are missing, which is {:1.2 f}% of
the total.'.format(len(smcRur), np.count_nonzero(smcRur), len(smcRur)-np.count_nonzero(smcRur),
100*(len(smcRur)-np.count_nonzero(smcRur))/len(smcRur)))

smcarrRur = np.array(smcRur)
smcarrRur = smcarrRur.astype('float')

for i in range(len(smcarrRur)):
if smcarrRur[i] == 0:
smcarrRur[i] = np.nan
else:
smcarrRur[i] = smcarrRur[i]

smcarrGeul = np.array(smcGeul)
smcarrGeul = smcarrRur.astype('float')

for i in range(len(smcarrGeul)):
if smcarrGeul[i] == 0:
smcarrGeul[i] = np.nan
else:
smcarrGeul[i] = smcarrGeul[i]

plt.plot(smcRur)
plt.plot(smcGeul)
```