LSTM Models Applied on Hydrological Time Series

And Their Potential Physical Implications

Bernhard Nornes Lotsberg

Thesis submitted for the degree of
Master in Computational Science: physics
60 credits

Department of Physics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2021

LSTM Models Applied on Hydrological Time Series

And Their Potential Physical Implications

Bernhard Nornes Lotsberg

LSTM Models Applied on Hydrological Time Series

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Today's process-driven hydrological models struggle to accurately model the complex large scale physical systems the field of catchment hydrology consists of. As others before us, we exploit newly released large-sample datasets that combine hydrological time series and static basin attributes. We show that LSTM models are able to generalize and make satisfactory predictions on ungauged basins, also when trained on two different datasets at the same time. An LSTM trained on CAMELS and CAMELS-GB datasets achieves a median validation NSE of 0.66 and 0.78 respectively, compared to the NWM, which scores a median of 0.55 on CAMELS alone. We currently know no publicly available NWM benchmarks performed on CAMELS-GB. In addition to this, we rank the basin attributes using permutation feature importance. The attributes deemed most important by the LSTM models are mostly attributes derived directly from the time series the model already has access to. This indicates that there is potential to improve the long term memory of our machine learning models. Our hope is that improving this in the future could lead to attribute rankings that indicate which attributes could be important for use in existing process-driven models. This either through adding new processes to existing models, or by guiding the tuning of model parameters through observable data instead of optimization schemes.

Acknowledgements

First of all I would like to thank my girlfriend for keeping me company during these last semesters. Writing a thesis in lockdown during a pandemic is not the most optimal scenario, but having here by my side writing her own thesis in the same situation has helped immensely.

I want to thank my main supervisor Simon Wolfgang Funke for our weekly discussions and his uplifting spirit. I also want to thank Matt Felix from Statkraft who has been heavily involved with this thesis.

The research presented in this paper has benefited from the Experimental Infrastructure for Exploration of Exascale Computing (eX3), which is financially supported by the Research Council of Norway under contract 270053.

Contents

Acronyms

CAMELS Catchment Attributes and MEteorology for Large-sample Studies.

CNN Convolutional Neural Network.

CV Cross Validation.

LSTM Long Short-Term Memory.

MSE Mean Squared Error.

NSE Nash–Sutcliffe model Efficiency coefficient.

NWM National Water Model.

RNN Recurrent Neural Network.

SAC-SMA SACramento Soil Moisture Accounting.

VIC Variable Infiltration Capacity.

WRF-Hydro Water Research and Forecasting model Hydrological modelling system.

List of Figures

List of Tables

Chapter 1

Introduction

Rainfall-runoff modelling is a very important problem for scientists and companies in the field of hydrology. Improving the prediction of runoff/streamflow based on historic data could lead to more accurate prediction of power generation in hydro power plants, flood forecasting and general water resource management.

Today there exists several models for this usage, some with less focus on physics: conceptual models, and some with a larger focus on physics: process-driven models. A drawback of these models is that they have many parameters that have to be calibrated for each basin individually [*Newman et al.*, 01 Aug. 2017; *Mendoza et al.*, 2015; *Kratzert et al.*, 2019a]. This leads to the models failing to generalize, as well as leading to the reduction of physical interpetability. Process-driven models with physical interpretations of their parameters have been shown to perform better once one forgoes the expected values of said parameters, and instead callibrate these as well [e.g. *Newman et al.*, 01 Aug. 2017; *Mendoza et al.*, 2015]. This is mostly due to the fact that the models are are not purely based on the underlying physics in a system, such as for conceptual models, and due to the otherwise excessive need to do on-site data collection for use in more heavily process-driven models.

With the explosive increase of observational data in recent years (mostly from improvement in sensor and satellite techniques) and the increase in computational power, new methods for streamflow forecasting can be explored. The new data could lessen the need to do on-site probing to model the complex physical system that is subsurface flow. Implementing new data in existing mdoels is hard, a simpler way to implement it is instead to use purely data-driven models. Machine learning models have shown potential for predicting on catchments described in the CAMELS dataset [*Kratzert et al.*, 2018; *Addor et al.*, 2017]. The potential for using this dataset in addition to several, newer datasets from other regions of the world to train generalizable machine learning models is higher. Being able to perform un-

gauged prediction (predictions on a basin without callibrating on it) could yield the benefit of getting approximate prediction of a catchment where no streamflow measurements have been performed.

The downside of using machine learning models (as well as other purely data-driven models) is that the models are less interpretable. Even conceptual models, being less physically based than process-driven models, are somewhat consistent with how a catchment works in the real world. This, for instance, makes it easier to discover when a model is predicting non-physicaly consistent results, as one can look at the outcome of each individual process in the model. For machine learning models, there is no such structure, making this a much harder task. Despite this, machine learning models can still be used to gain an understanding of which attributes contained in large scale datasets carry the most information, therefore narrowing down the process of using them to improve process-driven models.

1.1 Goals

There are two main goals in this thesis: Analysing static attributes to get a further indication of which (if any) attributes can be used to improving the understanding of the underlying physics in rainfall-runoff modelling, as well as discovering whether existing machine learning models can generalize across more than one dataset spanning several regions. Attribute importance ranking across several datasets is also of interest.

1.2 Our contribution

We show that our model performs better using the extra information in the static catchment features in the CAMELS-GB [*Coxon et al.*, 2020] dataset than if it were trained only on timeseries, as well as being able to perform well on ungauged basins.

We are able to do an analysis on the importance of these attributes and give a brief indication where to begin using these attributes to improve conceptual and process-driven models.

For further analysis we also provide the main bulk of the code used in this thesis as a python package released under the Apache 2.0 license. This code is originally based on the code released by *Kratzert et al.* [2019b].

1.3 Thesis structure

This thesis is divided into eight chapters:

1. The introduction. Here we explain the importance of this work and briefly summarize what has been done by others before us.

2. Theory. This chapter is divided into two parts:

   (a) Rainfall-runoff modelling. This section describes the relevant theory for understanding the physics captured in today's popular rainfall-runoff models. We give a brief overview of a popular conceptual model known as SACramento Soil Moisture Accounting (SAC-SMA), and the process driven model Variable Infiltration Capacity (VIC), as well as briefly mentioning the National Water Model (NWM), along with drawbacks and limitations of these models.

   (b) Machine learning. Here we try to give a overview of the concepts needed to understand the machine learning models we employ in this thesis.

3. Data. Here we give a brief explanation of how the CAMELS and CAMELS-GB datasets are structured, so that the Method and Results chapters are more easily followed.

4. Method. Here we describe in detail how we structure our models, what programming frameworks we use and how we analyse our results.

5. Results. Here we present our most interesting results and state initial observations.

6. Discussion. This chapter presents analysis of our results. We try to find potential ways to connect our results to the goals of this thesis.

7. Outlook. We briefly present a simple concept for a hybrid model, a potential physically based regularization scheme, as well as a simple CNN-based modification to our model structure.

8. Conclusion. A short summary of our most interesting findings along with what we believe are the next steps for further research.

In addition, the appendix contains documentation of the code used in this thesis.

Chapter 2

Theory

2.1 Rainfall-Runoff modelling

In this section we introduce and give brief explanations of two types of rainfall-runoff models. These two types are

1. Conceptual models: This type of model is designed to work in a similar manner to the system it is used to model, without being based on the physical laws actually governing said system. To properly model a system, a conceptual model usually has a large amount of parameters that need to be optimized.

2. Process-based models: These models are in a larger degree based on physical laws. They still have parameters, some of which have to be optimized, but they should ideally be given physical interpretations.

Because they attempt to predict runoff using physically modelled processes, process based models have been referred to as the models of choice when predicting runoff in ungauged (meaning basins where there is no streamflow data available for calibration) basins. *Kratzert et al.* [2019a] argues that this is not necessarily the case and data driven machine learning models can also give state of the art predictions. While this may be the case, the ideal scenario would be having a well performing process-based model. This is because all models are likely to have a selection of outlier basins where they perform exceptionally poorly. A model incorporating physics is in this case likely to be more interpretable, leading to the possibility of being able to know whether one can trust a specific prediction or not. For ungauged basins this is a clear benefit, as this means one can evaluate the plausibility of a result without prior data.

2.1.1 SACramento Soil Moisture Accounting

The SACramento Soil Moisture Accounting (SAC-SMA) rainfall-runoff model is a widely used hydrological and serves the purpose of being a conceptual

model in this thesis. Originally introduced in 1973 by *Burnash et al.* [1973], the model has since been modified and improved by several others [*Koren et al.*, 2014]. A thorough explanation of the model can be found in *Georgakakos* [1986].

The model focuses on conceptually modelling streamflow, using either real or modelled precipitation and evapotranspiration data as input. As well as needing evapotranspiration, the model also needs to be coupled with a snow (and frozen soil) model to be able to function in colder regions. The conceptual snow model known as Snow-17 can be used for this [*Anderson*, 1973; *Newman et al.*, 01 Aug. 2017].

This is not derived from any physical equation, but can instead be described as a "bucket model", i.e. it treats different layers of soils as buckets with holes in the bottom. This allows some streamflow even when the soil layer is not saturated, but once saturated, most if not all water is allowed to flow past the layer. To correctly model a given basin, SAC-SMA needs parameters such as drainage coefficients to be optimized This behaviour is in many ways consistent with the real world. What differentiates SAC-SMA and other conceptual models from process-driven models is that the equations guiding the behaviour of the model aren't directly derived from physical laws. Conceptual models model the behaviour of a physical system by being designed to act similarly to said system, without actually being governed by the system's physical laws.

2.1.2 The Variable Infiltration Capacity model

The Variable Infiltration Capacity (VIC) model [*Liang et al.*, 1994] is a process driven model with emphasis on simulating the physical process of water running through soil.

The name stems from the equation for variable infiltration capacity

$$i = i_m \left[1 - (1 - A)^{1/b_i}\right] \tag{2.1}$$

where

$$i_m = (1 + b_i)\theta_s|z|. \tag{2.2}$$

[*Liang et al.*, 1996] The terms in (2.1) and (2.2) are as follows:

- i: Infiltration capacity [m]
- i_m: Maximum infiltration capacity
- A: Fraction of soil area where $i < i_m$
- b_i: Infiltration shape parameter. Often found using calibration.
- z: Soil depth.

- θ_s: Soil porosity. Related to maximum soil water content with $W^c = \theta_s|z|$

The model was created to be used in a GCM (General Circulation Model), but can also instead used for rainfall-runoff modelling using real-world data instead of the inputs from a circulation model [*Newman et al.*, 01 Aug. 2017]. The rainfall-runoff modelling aspect of the model is presented in short here for context. It works by splitting the catchment area into N land coverage categories and the ground (vertically) into several soil layers. One thin upper soil layer and two lower soil layers. This is an attempt at physically representing underground streamflow[1]

The runoff modelling is split into several parts. Some of which are:

1. Evapotranspiration.
2. Surface/direct runoff from bare soil.
3. Subsurface runoff/baseflow from bare soil.
4. Covered surfaces (Types of land cover, vegetation).
5. Snow and frozen soil representations.

The volumetric water content of soil layer i is defined as

$$\theta_i = \frac{1}{z_i - z_{i-1}} \int_{-z_i}^{-z_{i-1}} \theta dz. \tag{2.3}$$

To obtain the direct runoff Q_d [mm/day], Richard's equation, which is defined as

$$\frac{\partial \theta}{\partial t} = \frac{\partial}{\partial z}\left(D(\theta)\frac{\partial \theta}{\partial z}\right) + \frac{\partial K(\theta)}{\partial z} \tag{2.4}$$

where $D(\theta)$ is the soil water diffusivity and $K(\theta)$ is hydraulic conductivity [mm/day], is used. Richard's equation describes the flow of water in unsaturated flow, and is the central physical law in VIC. The use of Richard's equation as a physical basis for modelling is what separates VIC, being a process-driven model, from conceptual models such as SAC-SMA. Richard's equation is numerically integrated from $-z_2$ to 0[2] This gives

$$\frac{\partial \theta_2}{\partial t} z_2 = I - E - K(\theta)|_{-z_2}^{0} - D(\theta)\frac{\partial \theta}{\partial z}|_{-z_2}^{0} \tag{2.5}$$

[1] While *Liang et al.* [1994] originally used a two layer setup, later versions of the VIC model have introduced more layers. *Liang et al.* [1996] for instance showed that adding a smaller layer on top improved performance significantly. This three layer structure is also what has been used by *Newman et al.* [01 Aug. 2017], which is the reason we include VIC as an example of a process-driven model in this thesis.

[2] *Liang et al.* [1996] argues that calculating direct runoff separately for layer 1 and 2 is not beneficial.

[*Liang et al.*, 1996]. Here $I = P - Q_d$ is the infiltration rate and θ_2 according to (2.3) is $\theta_i = \frac{1}{z_2}\int_{-z_2}^{0}\theta dz$. Using this value for θ_i, direct runoff Q_d is then modelled as

$$Q_d\left[N+1\right]\cdot\Delta t = \begin{cases} P\cdot\Delta t - z_2(\theta_s-\theta_2), & i_0 + P\Delta t \geq i_m \\ P\cdot\Delta t - z_2(\theta_s-\theta_2) & \\ +z_2\theta_s\left[1-\frac{i_0+P\cdot\Delta t}{i_m}\right]^{l+b_i}, & i_0 + P\cdot\Delta t \leq i_m \end{cases} \tag{2.6}$$

where $\theta_s = \frac{W_c^i}{z_i}$ and $\theta_i = \frac{W_i}{z_i}$ [*Liang et al.*, 1996].

The baseflow Q_b is defined as

$$\frac{\partial\theta_3}{\partial t}\left(z_3 - z_2\right) = K(\theta)|_{-z_2} + D(\theta)\frac{\partial\theta}{\partial z}|_{-z_2} - Q_b \tag{2.7}$$

[*Liang et al.*, 1996].

Summing direct runoff and baseflow gives total runoff $Q = Q_d + Q_b$. For other ground covers than bare soil the model is slightly modified. We do not present the details of this in this thesis as the point isn't to implement VIC, but rather to give a general idea of the physical effects it employs to model rainfall-runoff processes. We therefore instead shift our focus to another challenge to the VIC: Frozen soil modelling.

The original VIC did not account properly for frozen soil, the effect of which is important in cold climates. The newest version of VIC [*Hamman et al.*, 2018] uses a modification from *Cherkauer and Lettenmaier* [1999] to model frozen soil.

The frozen soil model is based on the heat equation for frozen soil, which is defined as

$$C_s\frac{\partial T}{\partial t} = \frac{\partial}{\partial z}\left(\kappa\frac{\partial T}{\partial z}\right) + \rho_i L_f\left(\frac{\partial\theta_i}{\partial t}\right). \tag{2.8}$$

From (2.8), *Cherkauer and Lettenmaier* [1999] then uses an explicit first order finite difference scheme to solve this one dimensional partial differential equation, discretizing $\partial t \approx \Delta t$ and $\partial z \approx \Delta z_i$. For the top two layers it is then easy to rewrite the equation on numerical form as

$$\begin{aligned} C_s\frac{T_i^t - T_i^{t-1}}{\Delta t} = &\left(\frac{\kappa_{i+1}^t - \kappa_{t-1}^t}{\alpha}\right)\left(\frac{T_{t+1}^t - T_{t-1}^2}{\alpha}\right) \\ &+ \kappa_i^t\left[\frac{2T_i + 1^t + 2T_{t-1}^t - 4T_i^t - 2\gamma f'(z)}{\beta}\right] \\ &+ \rho_i L_f\frac{(\theta_i)_i^t - (\theta_i)_i^{t-1}}{\Delta t}. \end{aligned} \tag{2.9}$$

The terms in (2.8) and (2.9) are as follows:

- C_S: Soil volumetric heat capacity [J/(m^{3}K)]

- T: Temperature [K]
- z: Depth [m]
- κ: Soil thermal conductivity [W/(mK)]
- ρ_i: Ice density [kg/m^3]
- L_f: Enthalpy of fusion [J/kg]
- θ_i: Ice fraction in soil [-]. When $\theta_i = 1$ there is no frozen soil and the term $\rho_i L_f \left(\frac{\partial \theta_i}{\partial t}\right)$ in (2.8) is defined as zero (the gradient of 0 is 0).
- $\alpha = (\Delta z_1 + \Delta z_2)$
- $\beta = (\Delta z_1^2 + \Delta z_2^2)$
- $\gamma = (\Delta z_1 - \Delta z_2)$
- $f'(z) = (T_{i+1}^t - T_{i-1}^t)/\alpha$

Again we see the use of physical equations to model a phenomenon. If this were a conceptual model, it would instead try to model the behaviour of frozen soil using equations that act like frozen soil in the real world, without basing them directly on physical equations such as (2.8).

The unfrozen water content of layer i when ice is present is then described as

$$W_i = W_i^c \left[\left(\frac{1}{g\psi_e}\right)\left(\frac{L_f T}{T + 273.16 \text{ K}}\right)\right]^{-B_p}. \tag{2.10}$$

- ψ_e: Air entry potential [m]
- B_p: Pore-size distribution [-]
- g: Surface average gravitational acceleration [m/s^2]

Obtaining θ for use in (2.6) and (2.7) is then done using (2.10) for cases where frozen soil is present (though not for drainage).

While *Liang et al.* [1994] included a simple way to model snow, which treated snow coverage as a special land coverage feature where it was assumed that all the land was covered by a uniformly divided snow layer, many updates later down the line by users of VIC have added to and improved these representations. The reason we explain this model in such detail here is to show that while it generally seems to be physically motivated, it still has a non-trivial amount of parameters that need to be set. Many of these parameters are given an a-priori physical meaning, but experiments have shown that the model actually performs significantly better when more parameters are optimized instead of using physically motivated

values [*Newman et al.*, 01 Aug. 2017; *Mendoza et al.*, 2015]. This could imply that the model does not account for several important physical phenomena. Speaking in the language of machine learning, the fact that adding more "learnable" parameters improves performance implies a lack of model complexity, "complexity" in this scenario meaning the ability of a model to fit a given training dataset.

Other, more widely used and developed process-driven also exist. One of which is the National Water Model (NWM). The NWM is a coupled meteorological forecast model, the hydrological component of which is known as the Weather Research and Forecasting Hydrological modelling system (WRF-Hydro) [*Gochis and Chen*, 2003; *Gochis et al.*, 2020]. In this thesis we refer to WRF-Hydro when mentioning the NWM. As a process-driven model, the NWM is more constrained than SAC-SMA, but known to better generalize than conceptual models.

2.1.3 Drawbacks

The reason we briefly describe several traditional models in this chapter is to give proper motivation for why it is of interest to see whether the information contained in easily obtainable new data has the potential to improve our physical understanding of the relationship between rainfall and runoff. It has been shown by *Kratzert et al.* [2019a] that one can, using machine learning, extract useful information from these attributes. Understanding the limitations of process driven models is important if one is to try finding a meaningful relationship between the physics not currently described fully in today's process-driven models and new types of data.

2.2 Machine Learning

We give a brief explanation of the basics in Machine Learning here for better context before elaborating on the LSTM model central to this thesis. Machine Learning is a type of frequentist approach to statistical analysis where one creates a statistical model, often with several million parameters and finds the value of each parameter that makes the model approximate the data in the most accurate manner. How to find these parameters and how they are used differ for each model type.

2.2.1 Linear regression

In the simple case of the Ordinary Least Squares (OLS) model we have a model on the form

$$\hat{\mathbf{y}} = \beta \mathbf{X} \tag{2.11}$$

This assumes that the outcome $\hat{\mathbf{y}}$ can we represented as a linear combination of some fitted parameters β and the input features $\mathbf{X}$. The goal here is then

to find the minimum of the mean squared error (MSE) of this. The MSE is defined as

$$MSE = |\mathbf{y} - \hat{\mathbf{y}}|^2 \tag{2.12}$$

Here $\mathbf{y}$ is the observed outcome, in many cases called the ground truth. $\hat{\mathbf{y}}$ is the prediction made by (2.11). The goal is to find the β that minimizes (2.12). For this there is an analytical solution as long as the matrix in (2.11) is reversible. In other words: This can be solved analytically as long as there are more data points than there are variables (features, inputs). The solution to the equation can be written as

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{2.13}$$

2.2.2 Bias-Variance trade-off

When training any kind of machine learning model, one usually divides the data into at least two parts: The training dataset and the testing dataset. The training dataset is for training a given model, while the testing dataset is to be kept separate from the training process so as to not make the performance metrics of the model too optimistic. By "optimistic" what is meant is that the error, for scalar values (2.12) is lower on the data the model is trained on than on data the model has not seen under training. The function that is minimized under machine learning is called a cost function, and while (2.12) is very commonly used for scalar outcomes, there exist many other cost functions all with different characteristics. To explain why this is important we need to have a quick look at what is known as the bias variance trade-off. In the case of the OLS model the MSE can be rewritten into four parts:

$$MSE = \text{Bias}^2 + \text{Variance} + \sigma^2 \tag{2.14}$$

For a full derivation of this, see *Vijayakumar* [2007]. When selecting and configuring machine learning models this trade-off is essential. The following is a qualitative explanation of what each term in (2.14) represents:

- Bias: The bias is the part of the error that comes from a model's lack of complexity. If one were to try and represent a non-linear system on the form of (2.11) for instance one would struggle to model the more complex interactions between input and outcome.

- Variance: In many ways this error is the opposite of bias. It comes from a given model having too much complexity. This could come from the model having too many parameters to train compared to how much data is available for training.

- σ^2: This is known as the irreducible error. It is the inherent error in the data that is used for training. The model cannot reduce the

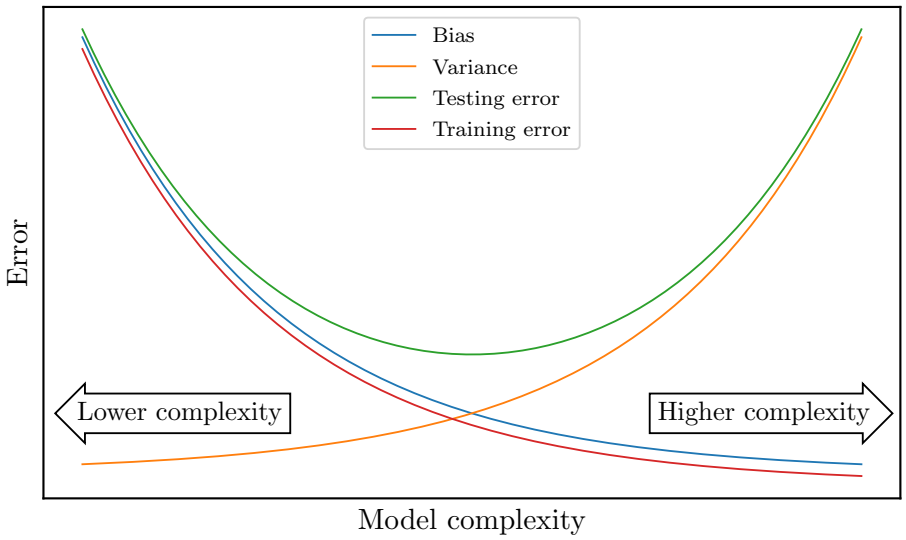

Figure 2.1: Illustration of the bias-variance tradeoff. The concept and function form is in part inspired by Figure 2.11 in *Hastie et al.* [2009]. For real world data the effect is not guaranteed to be this drastic.

error below this value as it is independent from the model. To reduce this error one would have to gather more accurate data using better instruments for instance.

The aim of training a machine learning model is to find the best tradeoff between model complexity and stability in search of the minimum of (2.14). This concept also generalizes to other types of modelling than machine learning. In *Newman et al.* [01 Aug. 2017] the concept of model flexibility is mentioned. This is analogous to a model with high variance, making it able to predict well in more scenarios, but possibly losing performance (overfitting) and interpretability if taken too far. In traditional rainfall-runoff modelling we could call a purely process-driven model biased, while a more data driven conceptual model would have higher variance.

The concept of the bias-variance tradeoff concept is illustrated in Figure 2.1. When a model is to simple it fails to properly capture the behavior of a given system and the error increases. When a model is too complex it starts fitting the train set perfectly and loses all generalizability, the test error therefore increases.

There are many ways to combat this issue. The most obvious solution is to increase the amount of data a model is trained on. This reduces the amount of model parameters compared to the model of data points and makes the model less likely to overfit. Another way is to forcefully increase

the bias of a model with high complexity through a process known as regularization. Regularization comes in many forms, often specific to the models that are being regularized.

2.2.3 Gradient Descent

While the simple case of linear regression has an easily attainable analytical solution as seen in (2.13), more advanced models may not. A generic computational optimization algorithm known as Gradient Descent can therefore be beneficial to employ in such a case [*Hastie et al.*, 2009] [*Géron*, 2019]. The fundamental concept of Gradient Descent is to calculate the gradient $\Delta F(\mathbf{x})$ of the loss function $F(\mathbf{x})$. In the case of regression this function would be the MSE as described in (2.12) while the variable $\mathbf{x}$ would be the regression coefficients β. After calculating $\Delta F(\mathbf{x})$ one then does a gradient descent step, which is defined as

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \lambda \Delta_x F(\mathbf{x_i}). \tag{2.15}$$

Here the subscript i denotes the epoch. An epoch is defined as the point when the optimization algorithm has seen all the data in your training set once. λ is called the learning rate and is known as a hyperparameter. A hyperparameter is a model parameter that is not trained in the training process and has to be optimized in a different way. More on this in chapter **??** A problem with (2.15) is that one is never guaranteed to find the absolute minimum of $F(\mathbf{x})$, the algorithm often converges at local minima [*Hastie et al.*, 2009]. The solution to this is to introduce stochasticity to the optimization algorithm. Stochastic Gradient Descent is a modification of (2.15) which iterates over minibatches of your dataset instead of the entire dataset at once. This has two main benefits:

1. It adds stochasticity by shuffling the batches for each epoch.

2. It significantly reduces the amount of RAM needed to run the calculations on a computer.

Introducing minibatches yields another hyperparameter in form of the batch size. This is the amount of datapoints to be used in each iterations. An iteration is defined as when the optimizer has seen all data points in a minibatch, and there are therefore several iterations in an epoch. Adding more complexity to Stochastic Gradient Descent (SGD) we get to the optimizer known as Adaptive Moment EstimAtioN (Adam) [*Kingma and Ba*, 2014]. The ADAM optimizer is an optimizer which uses minibatches and also what is known as a momentum based approach. As a slight simplification from what is written in *Hjorth-Jensen* [2020] the ADAM algorithm in terms of

equations looks like:

$$\mathbf{g}_i = \Delta_x E(\boldsymbol{x}) \tag{2.16}$$

$$\boldsymbol{m}_i = \frac{\beta_1 \boldsymbol{m}_{i-1} + (1-\beta_1)\boldsymbol{g}_i}{1-\beta_1^i} \tag{2.17}$$

$$\boldsymbol{s}_i = \frac{\beta_2 \boldsymbol{s}_{i-1} + (1-\beta_2)\boldsymbol{g}_i^2}{1-\beta_2^i} \tag{2.18}$$

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \eta_i \frac{\boldsymbol{m}_i}{\sqrt{\boldsymbol{s}_i} + \epsilon} \tag{2.19}$$

Following the same naming scheme as (2.15), $\boldsymbol{x}$ is the learning parameter of the model and the subscript i denotes the epoch (except for $\beta_{1/2}^i$, where it means power). β_1 and β_2 are two constants usually set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$. ϵ is a regularization constant to avoid numerical instability in the fraction and is usually $\epsilon = 1E-8$[3]. $\boldsymbol{m}$ and $\boldsymbol{s}$ are known as the first and second momentum of the gradient $\boldsymbol{g}$ respectively. The learning rate η_i is here subscripted with the epoch i. This is to allow for non-constant learning rates which is often beneficial The benefits of the ADAM optimizer is that it has low memory requirement and it calculates individual learning rates for for different parameters of a learning algorithm. This makes the learning rate λ a bit more ambiguous as it is weighed differently for each parameter.

2.2.4 Neural Networks

Previously we used Ordinary Least Squares to introduce simple concepts in Machine Learning. As the model we use to do our analysis in this thesis is a type of Recurrent Neural Network, we now give a short introduction to ordinary neural networks before working us up to describing more advanced models like Recurrent Neural Networks (RNNs). A neural network is a non-linear machine learning model that can be used for both regression and classification. In this thesis we focus on the regression case and will therefore stick to describing that. The concept of the neural network was first described in *Rosenblatt* [1958]. Mathematically, a neural network can be written in this form:

$$y_i^l = f^l \left(\sum_{j=1}^{N_{l-1}} \omega_{ij}^l y_j^{l-1} + b_i^l \right) \text{[4]} \tag{2.20}$$

[3] These values for β_1, β_2 and ϵ are mentioned in *Kingma and Ba* [2014] and seem to be used in practice in most cases. They are also the default values of the ADAM implementation in the Machine Learning Framework Pytorch [*Paszke et al.*, 2019].

[4] While the concept is cited to *Rosenblatt* [1958], this equation was taken from Morten Hjorth-Jensen's great lecture notes *Hjorth-Jensen* [2020] from the Universuty of Oslo course Fys-Stk4155 as this is the author's favorite mathematical description of a neural network

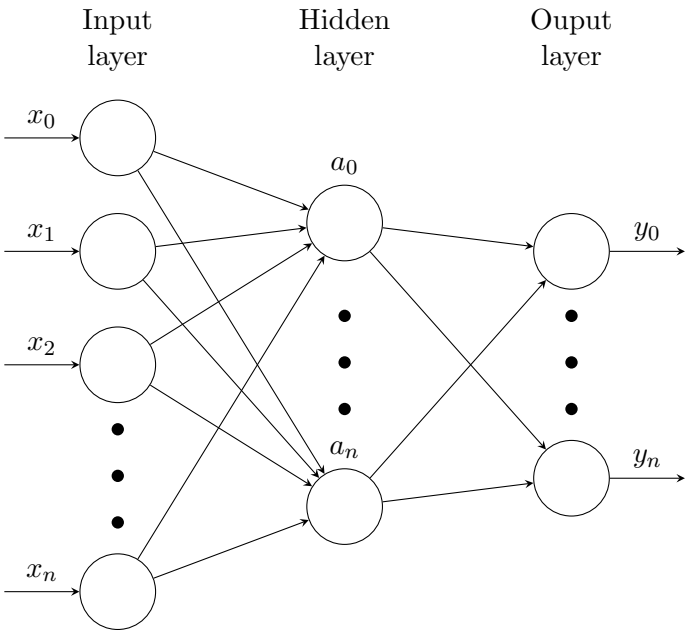

Figure 2.2: A multilayer perceptron model as described in *Rosenblatt* [1958]. Figure taken and modified from *Wibrow* [2014].

There are a lot of variables, subscripts and superscripts here, going through them systematically so as to not overwhelm the reader:

- l denotes what layer in the neural network we are in. Look at Figure 2.2 to understand what is meant by a layer.
- i denotes what "neuron" in layer l we are looking at.
- y_i^l means the output of neuron i in layer l. At the output layer $l = L$, it denotes the model output.
- f^l is the activation function in layer l. An activation function is a function that is used to make the neural network non-linear. Some common activation functions are the Sigmoid function (2.21) and tanh. For the output layer $l = L$ the activation function is often different depending on whether we are doing classification or regression. For regression f^L is often just $f^L(x) = x$.
- ω_{ij}^l is the weight corresponding to the output y_j^{l-1} from neuron j in layer $l-1$ when sent to neuron i in layer l
- b_i^l is known the bias term. It is a constant that is added to all inputs in neuron i in layer l.

- The input to f^l (the weighted sum plus the bias) can also be denoted as z_j^l

Figure 2.2 gives a visual representation of (2.20). The parameters b and ω are trained using an optimizer. In the case of this thesis that would be the ADAM optimizer. To get gradients to use in the optimizer an algorithm known as backpropagation is used. Backpropagation is an algorithm that takes the gradient at the output layer L, which usually is easily derivable and uses the chain rule to find the gradient of the loss function with respect to all the weights and biases in the network. We mention the sigmoid function, it is defined as

$$\sigma(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{x}}} \tag{2.21}$$

For a normal feed forward neural network, the backpropagation algorithm can be written as

$$\delta_j^L = \frac{\mathrm{d}f^L(z)}{\mathrm{d}z}\left(z_j^L\right)\frac{\partial C}{\partial y_j^L} \tag{2.22}$$

$$\delta_j^l = \sum_k \delta_k^{l+1}\omega_{kj}^{l+1}\frac{\mathrm{d}f^l(z)}{\mathrm{d}z}\left(z_j^l\right) \tag{2.23}$$

as shown by *Hjorth-Jensen* [2020]. The first equation (2.22) is for the special case of the output layer, where $l = L$. This needs to be calculated first for us to be able to use (2.23) to calculate δ_j^l for the neurons in the rest of the layer. The term δ_j^l is then used to get gradients for the weights ω_j^l and biases b_j^l in this manner:

$$\frac{\partial C}{\partial \omega_j^l} = \delta_j^l y_j^l \tag{2.24}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.25}$$

Applying an optimizer step on all weights and biases using 2.24 and 2.25 will then ideally make the model reduce the cost function of the training data. There is more to this than that though, hyperparameters, model complexity (often referring to the amount of parameters. In the case of a neural network that would be the amount of nodes and layers) dictate how well the model can fit the training data. As we discuss in chapter 2.2.2 we do not always want the model to perfectly predict the training data as that can make the model perform worse on data is has not seen during training. There is also the problem of the optimization algorithm not necessarily hitting a global minimum then converging. The parameter-loss space is a complex high dimensional space often with infinitely many local minima that can lead the model to a worse fit than would be possible otherwise [*Hastie et al.*, 2009].

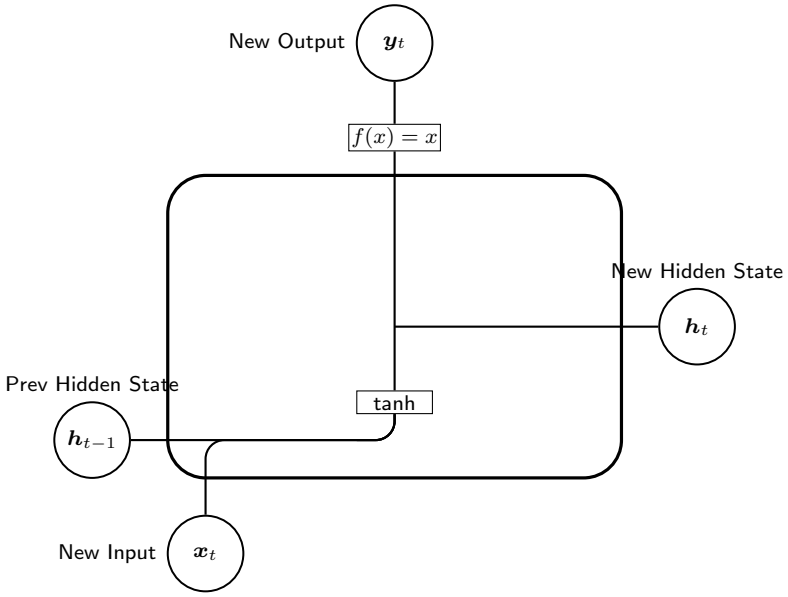

Figure 2.3: Illustration of an RNN cell. The squared tanh indicates a neural network layer with tanh as the activation function. Lines meeting indicate concatenation while lines splitting means copying. The squared $f(x) = x$ is a neural network layer with no activation function. The Tikz code for creating this figure is released under the permissive Beerware license and is a modified version of the figure in *Leon* [2018].

2.2.5 Recurrent Neural Networks

A normal feed forward neural network is only sufficient at learning the relationship between non-structured inputs and outputs. This means that it is not well suited for image analysis, where each pixel's position relative to other pixels is important. It also means that it is not sufficient for time series data where you ideally want a model than can sequentially run through the data and use information about the past to better predict the future.

A Recurrent Neural Network (RNN) is a neural network that takes timed inputs sequentially, changing modifying the "state" of the network while using the same parameters for all future time-steps. In simple terms the algorithm goes like this:

1. The model takes an input $\boldsymbol{x_t}$
2. Using trained parameters it updates a hidden state vector $\boldsymbol{h_t}$
3. The model gives an output based on the parameters and the hidden state $\boldsymbol{y_t}$.

For an ordinary RNN the state updates and outputs are defined as: [5]

$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{\omega}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{\omega}_{xh}\boldsymbol{x}_t + \boldsymbol{b}_h\right) \tag{2.26}$$

$$\boldsymbol{y}_t = \boldsymbol{\omega}_{hy}\boldsymbol{h}_t + \boldsymbol{b}_y \tag{2.27}$$

Here the variables are as follows:

- $\boldsymbol{y}_t$: The output vector (remember that there can be several output time series) at time step t.
- $\boldsymbol{h}_t$: The hidden state vector at time step t.
- $\boldsymbol{x}_t$: The input vector at time step t.
- $\boldsymbol{\omega}_{hh}$: The weight vector used for the previous hidden state to affect the new.
- $\boldsymbol{\omega}_{xh}$: The weight vector that decides how much the input $\boldsymbol{x}_t$ affects the state of the RNN.
- $\boldsymbol{\omega}_{hy}$: The weight vector that decides how much the hidden state h_t affects the output y_t.
- $\boldsymbol{b}_h$: A bias term that adds constant change to the hidden state.
- $\boldsymbol{b}_y$: A bias term used when calculating the next time step y_t.

In Figure 2.3 we see that the output at time step t is represented as equal to the hidden state fed through a neural network layer with no activation function. This is what is done in (2.27).

The backpropagation algorithm for an RNN is quite a bit different although the underlying concept is the same. We argue it is not necessary to include the algorithm in this thesis, but if the reader is interested they should read this source: *Werbos* [1990]. The important takeaway from the backpropagation algorithm used for RNNs (called Backpropagation Through Time), is that it uses high amounts of memory because it needs to store values for every time step to be able to get proper gradients for the model parameters. Because of this, it is common to instead use an algorithm called Trunctated Backpropagation Through Time, where one stops saving values after a pre-defined time sequence. This saves memory but makes the model unable to learn dependencies further in time than the cutoff. It also leads to the introduction of a new hyperparameter which we will refer to as the **sequence length**.

[5] Note that this describes only a single RNN cell. When dealing with RNNs we often use the term "cell" instead of "layer". There is nothing stopping us from creating a model architecture with multiple RNN cells. Also, keep in mind that there are many ways of sending in inputs and getting out outputs from RNN cells. Here we only present what is relevant for the model setup in this thesis.

2.2.6 Long Short-Term Memory

The Long Short-Term Memory (LSTM) model is the type of RNN we employ in all our experiments in this thesis work. In this section we therefore also spend some time arguing why the way an LSTM model works fits the Physics we wish to simulate and is not just chosen by random.

The LSTM is a more advanced type of recurrent neural network that exists because of a major drawback with the ordinary RNN. An RNN as described in the previous section struggles to learn long-term dependencies as the Backpropagation Through Time algorithm starts to downplay the importance of previous time-steps the further one advances in time. The fenomenon that causes this is known as the vanishing gradient problem [*Bengio et al.*, 1994] [*Graves*, 2012] [6]. The LSTM model is designed to fix this, hence the name. This is the reason we use the LSTM model and not a normal RNN in this thesis even though an RNN would be less computationally expensive to train, many of the dependencies in traditional rainfall-runoff modelling are long term. The most obvious case is the modelling of snow. Not the melting itself, but for the model to remember that snow accumulation can lead to lower discharge when it happens and more discharge once the snow starts to melt.

As opposed to the case of the vanilla RNN, trunctating the Backpropagation Through Time algorithm actually doesn't break long-term dependencies in an LSTM as the model architecture itself is designed to keep track of these [*Graves*, 2012]. This means that the LSTM has two clear advantages:

1. An LSTM does not suffer from the same vanishing gradient problem that the RNN does.

2. An LSTM is less sensitive to trunctating the Backpropagation Through Time algorithm.

To understand how an ordinary LSTM works we need to have a look at Figure 2.4. In the figure lines meeting means concatenation, the boxes represent neural network layers. The boxed σ for instance indicates a neural network layer with the Sigmoid function as the activation function. Circles indicate a pointwise operation. The biggest mathematical difference between a normal RNN and an LSTM is the introduction of the cell state $\boldsymbol{c}_t$. The cell state is only affected twice per time step by relatively simple operations (pointwise multiplication and addition). This ensures that the gradients related to the cell state get simplified and therefore suffer less from the vanishing gradient effect that hinders ordinary RNNs from learning long

[6] Mentions of vanishing and exploding gradients are common in Machine Learning and are not exclusive to Recurrent Neural Networks. Historically the popularity of activation functions in normal neural networks has also been dictated by these issues. The Sigmoid function (2.21) can lead to vanishing gradients as the gradient approaches 0 when $x \to \infty$.

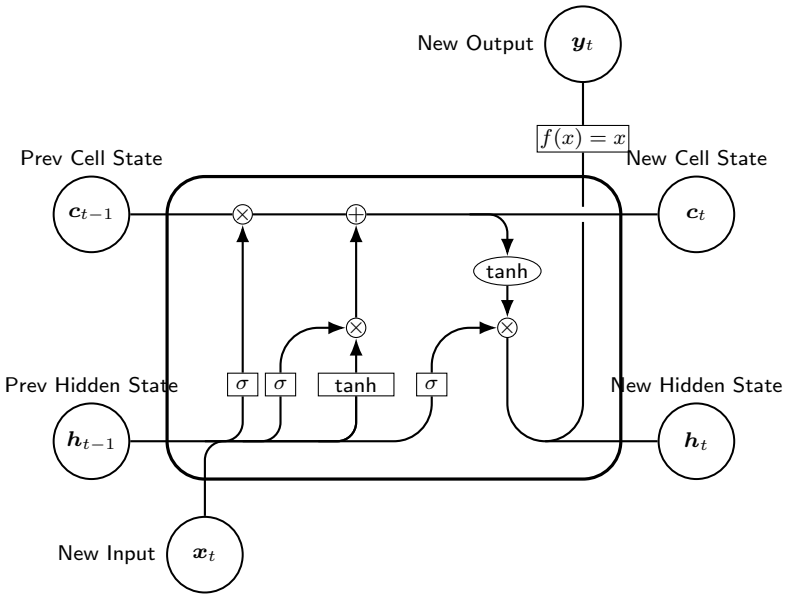

Figure 2.4: A single basic LSTM cell as proposed in *Hochreiter and Schmidhuber* [1997]. Visually we see the point of the Cell State C here is to not retain more information by being affected less over time than the hidden state h. This is how the LSTM avoids the gradient issues regarding long term dependencies that the RNN suffers from.The squared tanh indicates a neural network layer with tanh as the activation function, *sigma* indicated a sigmoid activation function and$f(x) = x$ indicates no activation function. The Tikz code for creating this figure is released under the permissive Beerware license and is a modified version of the figure in *Leon* [2018].

term dependencies. The three connections between the hidden state $\boldsymbol{h}$ and cell state $\boldsymbol{c}$ are often called "gates".

1. The first (from left to right) is often called the "forget gate". This is because $\sigma(x) \in \langle 0, 1 \rangle$, making the forget gate either not affect the hidden state at all or lower the values in the $\boldsymbol{c}_{t-1}$-vector.

2. The second gate is called the "input gate". This is where new information is added to the cell state through multiplying the output from two neural network layers, one with Sigmoid activation and one with tanh activation and then adding that product to the cell state.

3. The third gate is called the "output gate" and it dictates the flow of information from the cell state to the hidden state. It is defined as the pointwise product of $\tanh \boldsymbol{c}_t$ and the output of the concatenated previous hidden state and current input $[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]$. The output of this gate is both used as the updated hidden state $\boldsymbol{h}_t$ and as the output[7].

Now that we have a qualitative explanation of the LSTM model, we lay it down in mathematical terms as well:

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{\omega}_f \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f\right) \tag{2.28}$$

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{\omega}_i [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i\right) \circ \tanh\left(\boldsymbol{\omega}_{ii} [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_{ii}\right) \tag{2.29}$$

$$\boldsymbol{c}_t = \boldsymbol{c}_{t-1} \circ \boldsymbol{f}_t + \boldsymbol{i}_t \tag{2.30}$$

$$\boldsymbol{h}_t = \sigma\left(\boldsymbol{\omega}_o [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o\right) \circ \tanh(\boldsymbol{c}_t) \tag{2.31}$$

[*Hochreiter and Schmidhuber*, 1997] [*Gers et al.*, 1999].

The terms are as follows:

- σ: The Sigmoid function (2.21).
- $\boldsymbol{f}_t$: Forget gate at time step t.
- $\boldsymbol{\omega}_i$ and $\boldsymbol{f}_i$: Neural network weights and bias in the forget gate.
- $\circ$: Hadamard product.
- $\boldsymbol{i}_t$: Input gate at time step t.
- $\boldsymbol{\omega}_i$ and $_i$: Neural network weights and bias in the Sigmoid activated neural network in the input gate.
- $\boldsymbol{\omega}_{ii}$ and $_{ii}$: Neural network weights and bias in the tanh activated neural network in the input gate.
- $\boldsymbol{c}_t$: Cell state at time step t.

[7] At least in the model configuration we use, there are many variants of the LSTM model [*Graves*, 2012].

- $\boldsymbol{h}_t$: Hidden state at time step t.
- $\boldsymbol{\omega}_o$ and $\boldsymbol{b}_o$: Neural network weights and bias in the output gate.
- $\boldsymbol{x}_t$: Input vector at time step t.

There are several ways to use an LSTM for prediction. The method relevant for this thesis is the one we present here. To predict the output $\boldsymbol{y}_t$ at time step t:

1. Take a subset of the input vector $\boldsymbol{x}_{t-\text{sequence length}:t}$.
2. Feed it through the LSTM cell, saving the hidden vector $\boldsymbol{h}_t$.
3. Feed $\boldsymbol{h}_t$ through an ordinary neural network layer. This is essentially to do a weighted sum of the elements in $\boldsymbol{h}_t$ so that we get an output vector of the same dimensions as the output $\boldsymbol{y}_t$. With only one output as with rainfall-runoff modeling this means summing all the values $\boldsymbol{h}_t$ into a single scalar y_t.

2.2.7 Implementing static attributes along with time series

The description of the LSTM model we give in the above section only contains information for how to implement time series, not static attributes that do not vary with time. There are multiple ways to implement this. An obvious choice here is to naively treat each static feature as a time series that does not change. This seems counter-intuitive as it makes the model not see any difference between time series and static features, but it is very easily implementable and memory-wize does not affect the training algorithm too much[8]. This way we train the exact same type of model and do not need to modify anything.

Recently a small modification to the LSTM model was proposed for this purpose: The Entity Aware LSTM (EA-LSTM) [*Kratzert et al.*, 2019b]. Mathematically, an EA-LSTM is very similar to an LSTM. We only need to change the input gate described in (2.29):

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{\omega}_i \boldsymbol{x}_s + \boldsymbol{b}_i\right) \circ \tanh\left(\boldsymbol{\omega}_{ii}\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_{ii}\right) \tag{2.32}$$

This way the part of the input gate is affected by the static features, making the static features able to affect the cell state. This makes the model able to change how it views long-term dependencies based on information from the static features. The paper indicates that this modified LSTM performs worse than the naive approach, going from a median NSE of 0.76 to 0.74 on the CAMELS dataset [*Addor et al.*, 2017]. This may not always be the case, though. The EA-LSTM has fewer model parameters than an ordinary LSTM and can therefore be described as less complex. This may lead to better generalization in some cases, as it could function as a form of regularization.

[8] Memory is often the bottleneck of training machine learning models.

2.2.8 Addressing common criticisms of Machine Learning

There are several common criticisms of machine learning and other data driven approaches to physical modelling. Two common ones are:

1. Machine learning models are hard to analyze as they essentially are just a collection of arbitrary parameters that are optimized based on a dataset. It is therefore difficult to say whether a machine learning model actually "learns" any physics[9].

2. Machine learning models are unable to include the a-priori knowledge already present in most scientific fields.

We believe both these criticisms are valid and need to be addressed. First off: While analysing machine learning models can be very difficult, that is not necessarily always the case. It depends greatly on how complex the model is how much different data the model uses among other things. A linear regression model is easy to interpret, but it becomes much less easy to interpret if it needs several thousand different inputs [*Hastie et al.*, 2009]. This is the reason we employ feature selection here in this thesis. Our goal is not to just get the highest performing model, but rather to use the model as a tool for physical discovery and as a supplement to traditional models. This also addresses point 2: We are in fact not looking to use LSTM models as replacements for physical models, but instead we look to explore options for improving traditional models. It is important to remember that the best performing classical rainfall-runoff models also include many optimized parameters that do not necessarily represent any physical phenomena. We agree with *Karpatne et al.* [2017] that the power of machine learning when it comes to leveraging data should be used in a way that improves our understanding of the underlying system we are modelling.

[9] Though this is a question of definition. What does "learning physics"actually mean?

Chapter 3

Data

In this chapter we briefly describe the two datasets used to get all results in this thesis. We introduce the Catchment Attributes and Meteorology for Large-sample Studies (CAMELS) dataset [*Addor et al.*, 2017] first, as it is the older dataset and is used as the basis for the analysis of *Kratzert et al.* [2018, 2019b, a], and then the CAMELS-GB dataset *Coxon et al.* [2020].

3.1 The CAMELS dataset

The CAMELS dataset [*Addor et al.*, 2017] is a dataset compiled from several previous hydrological datasets in an attempt to improve the availability of data for large-scale hydrological modelling. It includes data for 671 basins across the United States, with time series spanning from approximately the year of 1980 to 2008.

We choose to focus on the time series data used by *Kratzert et al.* [2019b]. This selection of data consists of an extension of the original Maurer dataset contained in CAMELS provided by *Kratzert* [2019b]. The time series provided by the extended Maurer dataset that we use as inputs are the following:

- Precipitation [mm/day]
- Shortwave Radiation [W/m^2]
- Maximum air temperature [C°]
- Minimum air temperature [C°]
- Water vapor pressure [Pa]

The streamflow data corresponding to the meteorological forcing data mentioned above stems from the United States Geological Survey (USGS). It contains streamflow for all the basins in the dataset.

In addition to time series, CAMELS contains several static basin attributes describing location and topography, climatic indices, hydrological

signatures, soil characteristics, geological characteristics and land cover characteristics for each basin. Attributes describing location and topography are derived from the USGS part of the dataset described in *Newman et al.* [2015], and are basin-wise averaged. The attributes of interest here are the topographic ones, as location is not beneficial to physical modelling [1]. The climatic indices are directly derived from the time series provided by Daymet, another forcing time series dataset contained in CAMELS. The climatic index `frac_snow` is derived from the fraction of rainfall during subzero temperature, for instance. The hydrological signatures are derived from the streamflow data provided by USGS. These attributes contain info such as the average daily discharge `q_mean`. The land cover statistics describe land cover such as the forest fraction and the dominant land cover class. There is only a land cover fraction for the dominant land cover class for each basin. The land cover data is derived from the Moderate Resolution Imaging Spectroradiometer (MODIS) data, except for `forest_frac`, which is derived from USGS data. MODIS data comes from satellites, making it possible to gather equivalent data from other regions of the world without having to perform on-site measurements. The soil characteristics describe the soil properties of each basin in the dataset. These attributes are derived from *Miller and White* [1998]. There are two attributes for soil depth. One is from the paper just mentioned, and one is defined differently and stems from *Pelletier et al.* [2016]. In this thesis we use the attribute `soil_depth_pelletier`, as it stems from the same data source as in CAMELS-GB [*Coxon et al.*, 2020]. For a complete list of attributes in CAMELS, see Table 1 - 6 in *Addor et al.* [2017]. All attributes used in this thesis are shown in the upcoming chapter, in Table 4.2.

Complementing the CAMELS dataset are several hydrological model benchmarks provided by *Kratzert* [2019a]. These are calibrated and run on the time series data contained in CAMELS or the data it is derived from. In this thesis we include the benchmark of the VIC model we briefly describe in Section *Liang et al.* [1994]. Additionally *Kratzert et al.* [2019a] used and provided a preprocessed benchmark of the NWM [*NOAA*, 2018; *Salas et al.*, 2018]. This benchmark is known as the NOAA (National Oceanic and Atmospheric Administration) NWM reanalysis. The benchmark is released with no license and can as of April 2021 be found at `https://registry.opendata.aws/nwm-archive/`. The preprocessed version is available on *Kratzert et al.* [2019a]'s Github page: `https://github.com/kratzert/lstm_for_pub/blob/master/data/nwm/nwm_daily.pkl`.

[1] Though it could in theory improve the performance of a statistical model

3.2 The CAMELS-GB dataset

As per the Open Government License v3 the dataset is distributed under, we are required to include the statement "Contains data supplied by Natural Environment Research Council." The CAMELS-GB dataset [*Coxon et al.*, 2020] is inspired by the CAMELS dataset and is aimed to be the equivalent to the CAMELS dataset for basins in Great Britain, as opposed to North American (United Stated) basins in the CAMELS dataset. The dataset is the same in size, also containing 671 basins. The time series span from 1970-2015, though, making it span roughly 15 more years than the CAMELS dataset.

The forcing time series we use in this thesis are

- Precipitation [mm/day]
- Average temperature [C°]
- Wind speed [m/s]
- Humidity [g/kg]
- Shortwave radiation [W/m^2]
- Longwave radiation [W/m^2]

In addition there are two more time series which we exclude that describe potential evapotranspiration for well-watered grass. We exclude these as they are derived features and not measured, and they would make the comparison to CAMELS more difficult.

The static attributes in CAMELS-GB are structured similarly to those in CAMELS. There are location and topographic attributes, which describe the geographical location of each basin, as well as the giving a description of the topography. As stated above, the geographical attributes are not of interest to a physical model, and are therefore not used here. The topography attributes are provided by *Morris et al.* [1990]. The climatic indices are derived from the forcing time series, similarly to CAMELS. They contain average precipitation, aridity, snow fraction and other attributes. Hydrologic signatures are derived in the same manner, being processed information derived from the streamflow time series for each basin. The land cover attributes are provided by *Rowland et al.* [2017]. The classes have been derived using a random forest classifier on satellite images, meaning they suffer from uncertainty from the classification and the original data source. As opposed to CAMELS, there is land cover percentage for each class, not just the dominant class in each basin. `root_depth` is, like in CAMELS, taken from *Pelletier et al.* [2016]. All other soil attributes are provided by *Hiederer* [2013a, b]. The soil attributes are limited to the top 1.3 meters of soil.

CAMELS-GB contains three attribute categories not present in CAMELS: hydrogeological attributes, hydrometry and discharge uncertainty, and human influence attributes. These three categories are not used in this thesis. For a complete list of all the static attributes in CAMELS-GB, see Table 2 in *Coxon et al.* [2020]. As stated above, all attributes used in this thesis are also shown in Table 4.2.

The meteorological time series and static attributes in CAMELS-GB are not all equivalent with CAMELS. This makes it difficult to combine the datasets for a universal model, as one has to remove and/or modify a significant amount of static attributes to get overlapping datasets.

As far as we know there is no currently available traditional model benchmark dataset akin to *Kratzert* [2019a] for CAMELS-GB. The creation of such a dataset would likely be a challenge because of the time series contained in CAMELS-GB. There is for instance not enough information available to run VIC (see section 2.1.2) on the dataset.

Chapter 4

Method

In this chapter we discuss the implementation of the LSTM models described in Chapter 2.2.6. The goal is to use the hydrological data from the datasets described in Chapter 4 to predict runoff. We describe how we preprocess and combine the datasets and how the predictions of these models are used to try and gain insight into what physical processes our models deem the most important.

4.1 Code available as Python package: CamelsML

Originally a fork of the code in *Kratzert et al.* [2019b] with a few modifications, the machine learning code of this thesis is now implemented to be a fully fledged Python package. As the original code it is forked from, it is released under the Apache 2.0 license and anyone is therefore free to modify and implement the code into their own experiments in the future. We dub this package CamelsML (CAMELS Machine Learning).

See Appendix A.1 for documentation on how to use the python package as well as a minimal running example.

4.2 Training algorithm

The training algorithm, excluding the mathematical details shown in Chapter 2.2.6 is shown here:

1. Split the training data basin-wise into five parts of equal size.
2. Repeat the following five times, each time using a different 1/5 of the split data as the validation set:
   (a) Initialize LSTM model with random weights and zero biases, except for the bias of the forget gate $\boldsymbol{b}_f$, which is initialized as $\boldsymbol{b}_f = \mathbf{5}$.

$\boldsymbol{x}_{i+b,t}$	$\boldsymbol{x}_{i+b,t+1}$	. . .	$\boldsymbol{x}_{i+b,t+s-1}$	$\boldsymbol{x}_{i+b,t+s}$
$\boldsymbol{x}_{i+b-1,t}$	$\boldsymbol{x}_{i+b-1,t+1}$	. . .	$\boldsymbol{x}_{i+b-1,t+s-1}$	$\boldsymbol{x}_{i+b-1,t+s}$

.
.
.

$\boldsymbol{x}_{i+1,t}$	$\boldsymbol{x}_{i+1,t+1}$	. . .	$\boldsymbol{x}_{i+1,t+s-1}$	$\boldsymbol{x}_{i+1,t+s}$
$\boldsymbol{x}_{i,t}$	$\boldsymbol{x}_{i,t+1}$	. . .	$\boldsymbol{x}_{i,t+s-1}$	$\boldsymbol{x}_{i,t+s}$

Figure 4.1: A mini batch. $\boldsymbol{x}_{i,t}$ represents the input parameters $\boldsymbol{x}$ at time step t for time series $i \in [0, b]$ where b is the batch size. A mini batch consists of $b \times t$ FP32 numbers.

(b) Split each training time seres into several parts, the length of which is decided by the sequence length variable s. A mini batch consists of a batch size b amount of these s long time series. This structure is shown in Figure 4.1.

(c) For each mini batch:

i. Use the model to predict the outcomes of each time series in the mini batch. This can be done in parallel.

ii. Use the average loss of all predictions in the mini batch to update the model parameters using ADAM (see (2.16-2.19)).

(d) Evaluate on the validation set without updating the model parameters.

We evaluate models using the Nash–Sutcliffe model efficiency coefficient (NSE) [*Nash and Sutcliffe*, 1970]. This metric is essentially a hydrological interpretation of the well known R^2 score. It is defined as

$$\text{NSE} = 1 - \frac{\sum_{t=0}^{T} \left(y^t - \hat{y}^t\right)}{\sum_{t=0}^{T} \left(y^t - \bar{\hat{y}}\right)}. \tag{4.1}$$

Here T is the amount of time steps in a time series, y^t is the observed runoff used as the ground truth at time step $t \in [0, T]$, $\hat{y}^t$ is the predicted runoff at time step $t \in [0, T]$ and $\bar{\hat{y}}$ is the average predicted runoff.

Kratzert et al. [2019b] argues that using MSE as the cost function is not ideal for generalized hydrological modelling, as errors on basins with higher

absolute runoff will contribute more to the total MSE than those with lower runoff. To circumvent this they use what they dub the NSE loss function. It is defined as

$$NSE^*_{\text{basin}} = \frac{|\boldsymbol{y}_{\text{basin}} - \hat{\boldsymbol{y}}_{\text{basin}}|^2}{(\sigma_{\text{basin}} + \varepsilon)^2}, \tag{4.2}$$

where $\boldsymbol{y}_{\text{basin}}$ is the observed runoff of a given basin, $\hat{\boldsymbol{y}}_{\text{basin}}$ is the predicted runoff of a given basin and σ_{basin} is the standard deviation of a given basin's observed runoff. ε is a very small number included for numerical stability.

As we employ cross validation, we end up with 5 different models for each training run. These models do not necessarily converge to the same model parameters as each other, making them potentially quite different from one another. To test the actual performance of a model (not just relative to other model configurations) we therefore need to train a new model with the same configuration (using the same features, hyperparameters, etc.) on the entire, undivided training set and test that model on the test set. This test result cannot be used to determine relative performance between different model configurations, but it can give an indication of the actual, real-world performance of a final selected model. Statistically, this is due to the fact that optimizing model configuration using the test set would mean overfitting on said test set. See *Hastie et al.* [2009] for a more detailed explanation.

For models used to validate transfer learning between CAMELS and CAMELS-GB, we cross validate on one dataset and use all five models from the cross validation to make ensemble predictions on the validation split of the other dataset. This way we can get more robust statistics also in these results, as *Kratzert et al.* [2019b] showed that there often is a non-trivial performance difference between a single model and a model ensemble. In our case we assume this effect to be even higher because of the nature of training and validating on different datasets.

Table 4.1: Table containing all models trained in this thesis along with their given labels and configuration. All models are trained with a sequence length of 270 days and are initiated with the seed 19970204. The attribute subsets ***a*-*e*** are shown in Table 4.2.

Label	attribute subset	batch size	dropout
$GB_{\text{ea-lstm, all}}$	***a***	1152	0
$GB_{\text{lstm, all}}$	***a***	1024	0
$GB_{\text{ea-lstm, chosen}}$	***b***	1536	0
$GB_{\text{lstm, chosen}}$	***b***	1280	0
GB_{none}	-	1280	0
US_{Kratzert}	***c***	1024	0.4
US_{none}	-	2048	0.4
$Mixed_{\text{ea-lstm}}$	***d***	1024	0.4
$Mixed_{\text{lstm}}$	***d***	1024	0.4
$Mixed_{\text{none}}$	-	1024	0.4
$Transfer_{\text{GB, ea-lstm}}$	***d***	1024	0.4
$Transfer_{\text{GB, lstm}}$	***d***	4096	0.4
$Transfer_{\text{GB, none}}$	-	1024	0.4
$Transfer_{\text{US, ea-lstm}}$	***d***	4096	0.4
$Transfer_{\text{US, lstm}}$	***d***	4096	0.4
$Transfer_{\text{US, none}}$	-	4096	0.4
$Transfer_{\text{US, ea-lstm 2}}$	***e***	2048	0.4

Table 4.2: Table containing all basin attribute subsets. Subset ***e*** is equal to subset ***d*** but without organic_perc and gvf_max.

a		***b***	***c***	***d***	***e***
num_reservoir	reservoir_cap	area	soil_depth_pelletier	soil_depth_pelletier	soil_depth_pelletier
dwood_perc	ewood_perc	elev_10	soil_depth_statsgo	frac_forest	frac_forest
grass_perc	shrub_perc	elev_50	soil_porosity	gvf_max	
crop_perc	urban_perc	elev_90	soil_conductivity	p_mean	p_mean
inwater_perc	bares_perc	dwood_perc	max_water_content	pet_mean	pet_mean
p_mean	pet_mean	ewood_perc	sand_frac	p_seasonality	p_seasonality
aridity	p_seasonality	grass_perc	silt_frac	frac_snow	frac_snow
frac_snow	high_prec_freq	shrub_perc	clay_frac	aridity	aridity
high_prec_dur	low_prec_freq	crop_perc	frac_forest	high_prec_frec	high_prec_freq
low_prec_dur	inter_high_perc	urban_perc	lai_max	high_prec_dur	high_prec_dur
inter_mod_perc	inter_low_perc	inwater_perc	lai_diff	low_prec_frec	low_prec_freq
frac_high_perc	frac_mod_perc	bares_perc	gvf_max	low_prec_dur	low_prec_dur
frac_low_perc	no_gw_perc	sand_perc	gvf_diff	area	area
low_nsig_perc	nsig_low_perc	silt_perc	p_mean	porosity_cosby	porosity_cosby
gauge_lat	gauge_lon	clay_perc	pet_mean	conductivity_cosby	conductivity_cosby
gauge_easting	gauge_northing	organic_perc	p_seasonality	sand_perc	sand_perc
gauge_elev	area	bulkdens	frac_snow	silt_perc	silt_perc
elev_min	elev_10	tawc	aridity	clay_perc	clay_perc
elev_50	elev_90	porosity_cosby	high_prec_frec	organic_perc	
elev_max	sand_perc	porosity_hypres	high_prec_dur		
silt_perc	clay_perc	conductivity_cosby	low_prec_frec		
organic_perc	bulkdens	conductivity_hypres	low_prec_dur		
tawc	porosity_cosby	root_depth	elev_mean		
porosity_hypres	conductivity_cosby	soil_depth_pelletier	slope_mean		
conductivity_hypres	root_depth	inter_high_perc	area_gages2		
soil_depth_pelletier		inter_high_perc	carbonate_rocks_frac		
		inter_mod_perc	geol_permeability		
		inter_low_perc			
		frac_high_perc			
		frac_mod_perc			
		frac_low_perc			
		no_gw_perc			
		low_nsig_perc			
		nsig_low_perc			

Table 4.1 shows all trained models from which results are presented in this thesis. The largest differences between these models are which basin attributes are included in the training process. Other relevant hyperparameters are set equal to the model configuration in *Kratzert et al.* [2019b]. For CAMELS-GB, the dates included in the training process are time daily time steps from October 10th 1971 to September 30th 2015. For CAMELS, the dates used are January 1st 1980 to December 31st 2008. The models in Table 4.1 are split into four categories, one for each experiment (hence the names GB, US, Mixed and Transfer). This way we can measure and analyse the performance of LSTM models on CAMELS-GB, CAMELS (recreating the results of *Kratzert et al.* [2019a] with a differently configured cross validation), Mixed (both CAMELS and CAMELS-GB at the same time) and Transfer (training on CAMELS-GB and predicting on CAMELS, and vice-versa). One should be able to reproduce the results of this work using CamelsML, Table 4.3, 4.1 and 4.2. Alternatively one could find all the needed model configurations along with seeds on the Github page of this thesis: `https://github.com/bernharl/Master-Thesis`. Doing exhaustive validation to decide which subset of attributes to use is not feasible with the amount of attributes present in the data of interest. What we instead do to end up with the subsets in Table 4.2 is a mix of a priori knowledge and validation: First we manually check which attributes to include based on perceived importance in accordance to known physical processes related to rainfall-runoff modelling. After training a model on a subset, we evaluate it using cross validation as mentioned earlier in this chapter. To give further context for the chosen subsets ***a*-*e*** we present this short summary:

- ***a***: This is a subset using all numerical static attributes in CAMELS-GB [*Coxon et al.*, 2020] that are not derived from the outcome (runoff).

- ***b***: This is a smaller subset of the static attributes contained in CAMELS-GB. This subset was created with emphasis on perceived importance with respect to known physical processes. As many process-driven models (see Section 2.1.2) have high emphasis on radiation (from vegetation), soil types and land cover we have included all attributes related to soil, water content, vegetation and general land cover. The geographical layout of a basin is also of interest and elevation attributes are therefore included. The inclusion of the feature p_mean (mean precipitation) is however not physically motivated and merely stems from the fact that it was recognized as one of the most important features in *Kratzert et al.* [2019b]'s analysis using LSTMs on CAMELS [*Addor et al.*, 2017]. Mean precipitation should in theory be information already given in the precipitation time series, but our models do not have access to the entire precipitation time series while training because of the limited sequence length.

- ***c***: This attribute selection is taken from *Kratzert et al.* [2019a] and is used to reproduce the results of said paper with a different cross validation setup to better fit with the rest of our analysis. *Kratzert et al.* [2019a] used 12-fold cross validation while we use the more commonly employed 5-fold cross validation.

- ***d***: This is an attribute selection used for training models on both CAMELS and CAMELS-GB at the same time in addition to transfer learning. The attribute names stated in Table 4.2 are based on the attribute names in CAMELS-GB. CAMELS and CAMELS-GB have different attributes and different names for the same attributes. Which attributes we deem to be equivalent are shown in Table 4.3.

- ***e***: The attributes organic_perc and gvf_max are excluded in this subset, otherwise it is identical to subset e. We exclude these two features because of uncertainty of whether our synthetic attribute creation works.

4.3 Preprocessing and combining datasets

In statistical models it is important to have all inputs and outputs in unit-less form. A common way to achieve this is to normalize each input feature and outcome feature [*Hastie et al.*, 2009]. We split the data basin-wise into train (75%) and test (25%) sets. Five-fold cross validation is used on the training set to evaluate model performance. For each fold in the cross validation run, the data is normalized based on the current training set, excluding the chosen validation set in each cross validation iteration. Mathematically this can be written as

$$\boldsymbol{a}_{\text{norm}} = \frac{\boldsymbol{a} - \bar{\boldsymbol{a}}_{\text{train}}}{\sigma_{\boldsymbol{a}_{\text{train}}}}. \tag{4.3}$$

Here $\boldsymbol{a}$ represents any variable, input or output, $\hat{\boldsymbol{a}}_{\text{train}}$ is the average of said variable in the train set and $\sigma_{\boldsymbol{a}_{\text{train}}}$ is the standard deviation of said variable in the train set. If $\boldsymbol{a}$ is a time series, the averaging is still done for all time steps in all basins in the training set, not individually per basin. This normalization is implemented in CamelsML by saving the standard deviations and averages of each feature in the training set to the disk and is implemented by us. *Kratzert et al.* [2019a] has likely also implemented this in a different way, but this code is not used here.

When using a combination of CAMELS and CAMELS-GB we are limited in both which time series and which basin attributes we can use. As seen in Chapter 3 there are only two overlapping time series: precipitation and shortwave radiation. CAMELS-GB only has average daily temperature, while CAMELS has both minimum and maximum temperature per day. To

Table 4.3: Timeseries and attributes in CAMELS and CAMELS-GB that we treat as equivalent. The names are taken directly from *Addor et al.* [2017] and *Coxon et al.* [2020].

Camels-US	Camels-GB
time series:	
prcp(mm/day) [mm/day]	precipitation [mm/day]
$\frac{\text{tmax}+\text{tmin}}{2}$ [K]	temperature [K]
srad(W/m2) [W/m^2]	shortwave_rad [W/m^2]
attributes:	
elev_mean [m]	elev_mean [m]
area_gages2 [km^2]	area [km^2]
p_mean [mm/day]	p_mean [mm/day]
pet_mean [mm/day]	pet_mean [mm/day]
p_seasonility	p_seasonality
frac_snow	frac_snow
high_prec_freq [days / yr]	high_prec_freq [days / yr]
high_prec_dur [days]	high_pref_dur [days]
low_prec_freq [days / yr]	low_prec_freq [days / yr]
low_prec_dur [days]	low_prec_dur [days]
aridity	aridity
forest_frac	dwood_frac + ewood_frac
root_depth_50 [m]	root_depth_50 [m]
soil_depth_pelletier [m]	soil_depth_pelletier [m]
soil_porosity	porosity_cosby
soil_conductivity [cm/h]	conductivity_cosby [cm/h]
sand_frac	sand_perc / 100
silt_frac	silt_perc / 100
clay_frac	clay_perc / 100
organic_frac	organic_perc / 100
gvf_max	$1 - \frac{\text{urban\_perc}+\text{inwater\_perc}}{100}$

include temperature as a feature we therefore make the assumption that

$$\bar{t}_{\text{daily}} \approx \frac{t_{\text{min, daily}} + t_{\text{max, daily}}}{2} \tag{4.4}$$

where $\hat{t}_{\text{daily}}$ is the day-averaged temperature, $t_{\text{min, daily}}$ is the day-minimum temperature and $t_{\text{max, daily}}$ is the day-maximum temperature. The assumption in (4.4) only holds when the daily temperature maxima and minima vary symmetrically around an equilibrium. Most of the basin attributes we include in the combined dataset have natural equivalents in both datasets, but there are two exceptions. We get an equivalent of the attribute forest_frac in CAMELS in CAMELS-GB by assuming that

$$\text{forest\_frac}_{\text{GB}} \approx \text{dwood\_frac} + \text{ewood\_frac}. \tag{4.5}$$

As described in *Coxon et al.* [2020] dwood_perc is the percentage of deciduous woodland and ewood_frac is the percentage of evergreen woodland. Our reasoning for (4.5) is that dwood_frac and ewood_frac are the only forest attributes in CAMELS-GB and it should therefore be safe to assume that adding these together yields the total percentage of wood cover.

For the CAMELS attribute gvf_max we make a less safe assumption. *Addor et al.* [2017] describes this attribute as "maximum monthly mean of the green vegetation fraction". It could then follow that the if we subtract all land covers without vegetation we end up with something similar. In CAMELS-GB the two land covers not covered in vegetation are urban_perc (percentage of suburban or urban land) and inwater_perc (the percentage covered by inland water). This yields

$$\text{gvf\_max}_{\text{GB}} \approx 1 - \frac{\text{urban\_perc} + \text{inwater\_perc}}{100}. \tag{4.6}$$

A full overview of attributes deemed equivalent in CAMELS and CAMELS-GB is shown in Table 4.3.

Figure 4.2 shows boxplots of the training data of all attributes in attribute subset ***d*** (see Table 4.2) for both CAMELS and CAMELS-GB. In each subplot CAMELS-GB is on the left and CAMELS is on the right. We see that most attributes have higher variance in CAMELS than in CAMELS-GB. Exceptions to this are organic_perc and gvf_max. We therefore include transfer learning models both with and without these two features. This also implies that the assumption made in (4.6) may not be of use.

4.4 Basin attribute ranking

One of the criticisms of machine learning models is that they are not easily interpretable. This is especially true if one wants to train a model on a dataset with an overwhelming amount of features. To interpret our models

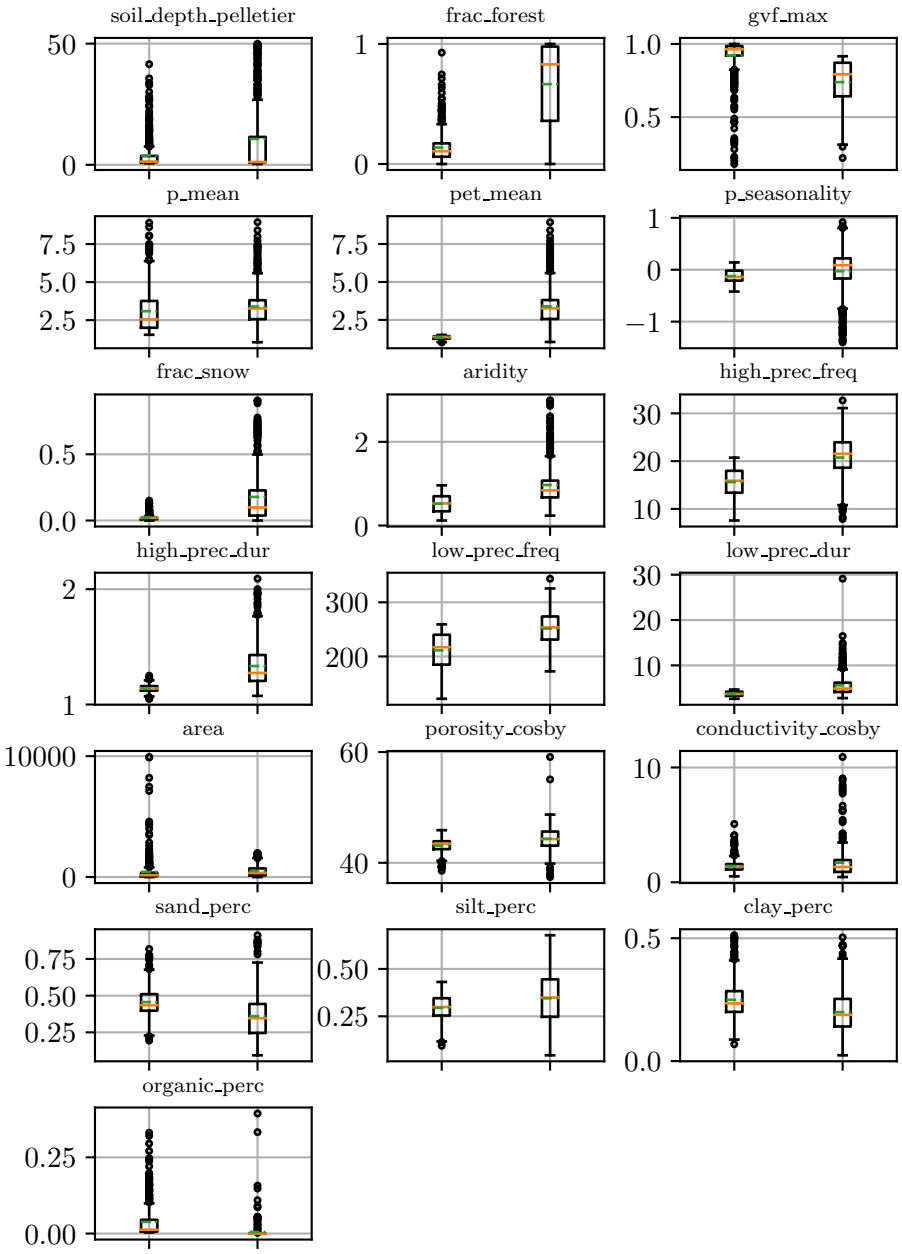

Figure 4.2: Boxplots of the basin attributes in CAMELS and CAMELS-GB compared. In each subplot CAMELS-GB is on the left, CAMELS is on the right. The basins included in this figure are the basins contained in the full training set used by all models in this analysis. The orange line indicated the median, the green dashed line indicates the average.

we implement a way to determine feature importance. There are many algorithms to determine feature importance, we choose to use the permutation feature importance algorithm for its simplicity, and ability to be used on any type of trained model. Given a feature j, the permutation importance i_j is equal to

$$i_j = s - \frac{1}{K}\sum_{k=1}^{K} s_{k,j} \tag{4.7}$$

[*Scikit-Learn developers*, 2020; *Breiman*, 2001]. Here K denotes how many permutations we average over for each feature, s is the model's score on the original data and $s_{k,j}$ is the score of permutation number k of the feature j. In essence (4.7) describes how much the performance of a model varies when scrambling the information contained in a feature, therefore explaining the importance of the feature according to the current model. It is then important to remember that this is not the true importance of the feature, only the importance the model thinks the feature has. The scoring method s can be any model scoring statistic, often the R^2. score for regression. In our case we employ (4.1) as the scoring metric and we only consider basins with an NSE of at least 0.5, as an NSE / R^2 score indicates that a model's prediction is at least as good as always predicting the average outcome.

A major problem for this method is that (4.7) could give unrealistically low significance to features that are highly correlated to other features. In this case the feature may very well be important, but the information contained in it is also contained in one or more other correlated features, meaning the model doesn't lose as much information as one may think. Essentially the rule of thumb is that the feature importance is the feature's importance as learned by the model, not the actual real world importance.

To validate that this method is implemented correctly and gives meaningful results in our case we train a model on CAMELS-GB that in addition to the attributes in attribute subset a of Table 4.2 has access to attributes directly derived from the runoff. This model cannot be used for anything but to validate the permutation importance algorithm, as it having access to parts of the outcome as an input compromises it's ability to do predictions on ungauged basins. The attributes in question are stated in Table 2 in *Coxon et al.* [2020] under the "Hydrologic signatures" class. One of them is for instance q_mean, which is the mean daily discharge (runoff).

4.5 Hardware

The models are run on three different hardware configurations. The important difference between these hardware configurations is the amount of available VRAM. As LSTM models are recurrent neural networks they are not as parallelizable as other machine learning models (see Section 2.2.5 and

2.2.6). This means that the only way for us to fully exploit an increase in VRAM is to parallelize data-wise. To do this we increase the batch size. The three hardware configurations are listed as follows [1]:

1. Nvidia®GTX™980 ti: This has 6 GB of VRAM. We find a batch size of $b \in 1024, 1536$ depending on the model size to be a sweet spot here.

2. Nvidia®GTX™1660 ti: This GPU is similar to the GTX 980 ti and has the same amount of VRAM. Therefore the same batch sizes apply here too.

3. Nvidia®Tesla™V100 (Provided by Simula's eX3 cluster): This has 32 GB of VRAM. We find a batch size of $b \in [2048, 4096]$ depending on the model size to be a sweet spot. Anything more leads to a downgrade in speed because of limitations in transferring data from the storage device to the GPU. This is still not ideal and leads to an approximate 30% utilization of the GPU.

We acknowledge that the inconsistent use of batch sizes across models may somewhat impact the model performance, as previously stated in Section 2.2.3. However, as long as the amount of mini batches is sufficiently smaller than the number of data points we still get an acceptable amount of stochasticity in the training process. As an increase in batch size is often directly correlated to the speed of training, the batch size usually set as high as the hardware supports. This is in most cases fine as long as the total size of training data is much larger than the available RAM (if training on CPU) or VRAM (if training on GPU). Still, in an ideal situation we would use the same hardware for all training runs.

[1] Note that we only state the graphics card used, as this is the only important difference. We run no calculations on CPU and do not use a significant amount of ordinary RAM.

Chapter 5

Results

In this chapter we present the results divided into five main sections:

1. Models trained on CAMELS-GB [*Coxon et al.*, 2020].
2. Models trained on CAMELS [*Addor et al.*, 2017] in addition to traditional models provided by *Kratzert* [2019a] and *NOAA* [2018].
3. Models trained on a dataset comprised of both CAMELS and CAMELS-GB.
4. Models trained on CAMELS and validated on CAMELS-GB (notated as US→GB) and vice-versa.
5. Models refit on the full train set and validated on the previously excluded test set.

5.1 Models trained on CAMELS-GB

In this section we present the results related to model selection and feature importance when training and predicting on CAMELS-GB [*Coxon et al.*, 2020]. To act as a proof of concept of the feature importance method described in Chapter 4.4, we also include results from an additional model which is trained on a superset of attribute subset $\boldsymbol{a}$ which includes attributes directly derived from the observed outcome. We label this model "Overfit model". By "attributes", we are referring to the static attributes included in CAMELS and CAMELS-GB.

5.1.1 Performance

Figure 5.1 shows the performance of $\text{GB}_{\text{lstm, all}}$, $\text{GB}_{\text{ea-lstm, all}}$, $\text{GB}_{\text{lstm, chosen}}$, $\text{GB}_{\text{ea-lstm, chosen}}$ and $\text{GB}_{\text{lstm, none}}$ trained on CAMELS-GB [*Coxon et al.*, 2020]. For an overview of these LSTM models, see Table 4.1. The overfit

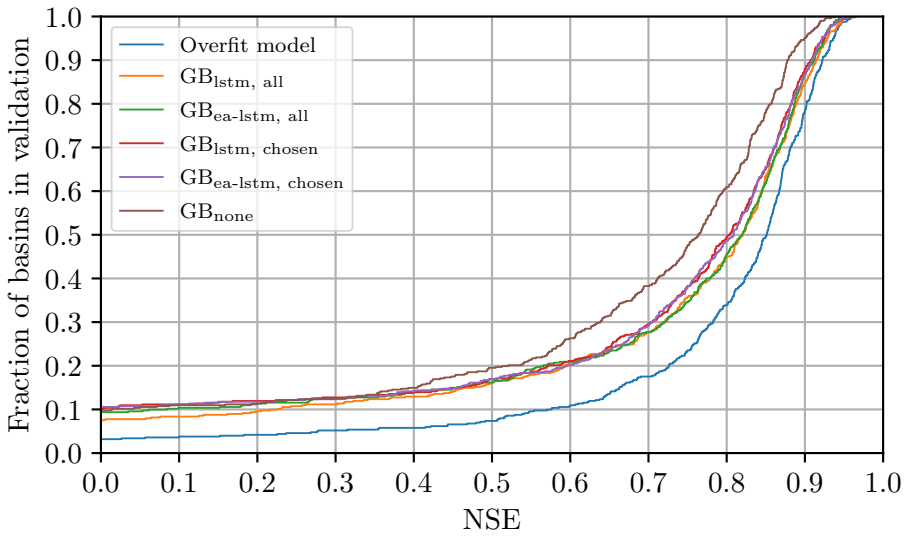

Figure 5.1: Cumulative distribution function of the NSE score of LSTM models trained on CAMELS-GB [*Coxon et al.*, 2020]. "Overfit model" is a model deliberately trained using static basin attributes derived from the runoff time series of the basins. The other models are described in Table 4.1.

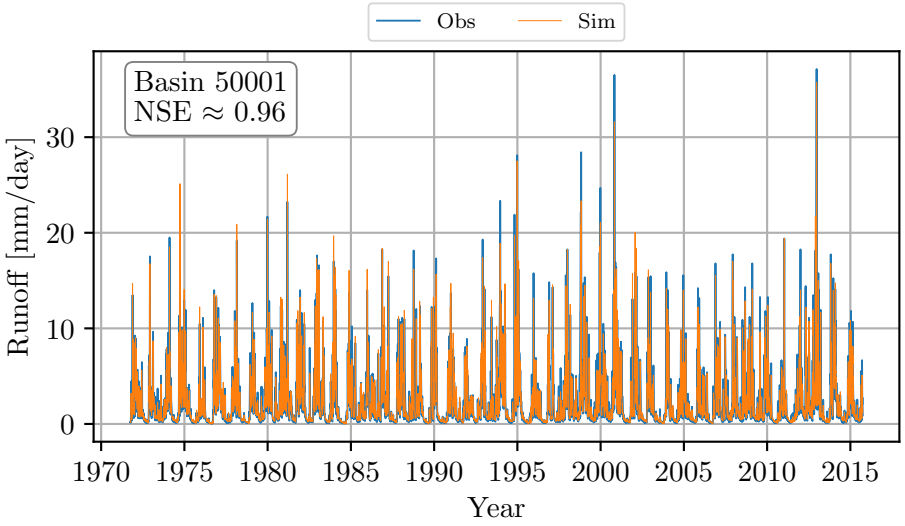

Figure 5.2: Highest scoring prediction on the validation set made using $\text{GB}_{\text{lstm, all}}$ compared to the observed outcome. "Obs" is observed runoff, "Sim" is predicted runoff. The basin data is taken from CAMELS-GB [*Coxon et al.*, 2020].

model is trained on attribute subset $\boldsymbol{a}$ in Table 4.2 in addition to attributes directly derived from the observed outcome. The overfit model significantly outperforms all other models. Of the other models there seems to be generally two levels of performance. The models using attribute subsets $\boldsymbol{a}$ and $\boldsymbol{b}$ perform similarly, although in the favour of attribute subset $\boldsymbol{a}$, while the models trained with no attributes in general perform worse. There seems to be little overall difference in performance between EA-LSTM and LSTM models, apart from $GB_{lstm,all}$ which performs with a negative NSE value for a smaller fraction of the basins. All other ungauged models perform with a negative NSE value for approximately 10% of the basins.

The highest scoring validation set prediction made using $\text{GB}_{\text{lstm, all}}$ is shown in Figure 5.2. It has an NSE score of ≈ 0.96. Most of the peaks are undervalued by the prediction made by the model.

5.1.2 Importance

Table 5.1 shows 5th, 25th, 50th, 75th and 95th percentiles of the importances of the top 20 basin attributes of the overfit model according to the permutation algorithm described in Chapter 4.4. We observe that the attribute with the highest importance in all quantiles is `Q95`, which is the 95th percentile runoff derived from the observed runoff time series. `Q95` has an

Table 5.1: Top 20 (ranked by median importance) attributes of the overfit model described earlier in this section according to the permutation feature importance algorithm. The columns are percentiles.

	95%	75%	Median	25%	5%
Q95	9.97	0.42	0.12	0.04	-0.01
baseflow_index_ceh	1.22	0.21	0.08	0.02	-0.01
p_mean	0.33	0.11	0.04	0.01	-0.02
aridity	0.10	0.04	0.01	0.00	-0.02
Q5	0.61	0.06	0.01	0.00	-0.03
p_seasonality	0.12	0.02	0.00	-0.00	-0.02
area	0.11	0.01	0.00	-0.00	-0.01
inwater_perc	0.12	0.02	0.00	-0.00	-0.02
elev_10	0.10	0.02	0.00	-0.00	-0.02
low_prec_dur	0.06	0.01	0.00	-0.00	-0.02
gauge_easting	0.05	0.01	0.00	-0.00	-0.03
low_prec_freq	0.04	0.01	0.00	-0.00	-0.02
elev_90	0.05	0.01	0.00	-0.00	-0.02
elev_50	0.09	0.01	0.00	-0.00	-0.02
conductivity_hypres	0.05	0.01	0.00	-0.00	-0.03
grass_perc	0.05	0.01	0.00	-0.00	-0.02
urban_perc	0.11	0.01	0.00	-0.00	-0.01
elev_max	0.04	0.01	0.00	-0.00	-0.02
high_prec_freq	0.04	0.01	0.00	-0.00	-0.02
pet_mean	0.05	0.01	0.00	-0.00	-0.02

Table 5.2: Top 20 (ranked by median importance) attributes of $GB_{lstm,\ all}$ according to the permutation algorithm. The columns are percentiles.

	95%	75%	Median	25%	5%
low_prec_dur	0.25	0.09	0.02	0.00	-0.04
low_prec_freq	0.35	0.06	0.02	0.00	-0.04
tawc	0.10	0.03	0.01	-0.00	-0.04
no_gw_perc	0.08	0.03	0.01	-0.00	-0.04
grass_perc	0.15	0.02	0.01	-0.00	-0.04
high_prec_freq	0.09	0.02	0.01	-0.00	-0.04
elev_90	0.11	0.02	0.01	-0.00	-0.03
elev_10	0.16	0.03	0.01	-0.00	-0.03
p_seasonality	0.14	0.03	0.00	-0.00	-0.03
aridity	0.09	0.02	0.00	-0.00	-0.03
root_depth	0.22	0.02	0.00	-0.00	-0.03
crop_perc	0.15	0.03	0.00	-0.00	-0.05
inwater_perc	0.28	0.03	0.00	-0.00	-0.03
frac_snow	0.18	0.02	0.00	-0.00	-0.02
elev_50	0.12	0.02	0.00	-0.00	-0.04
p_mean	0.09	0.02	0.00	-0.00	-0.03
conductivity_hypres	0.15	0.02	0.00	-0.00	-0.04
area	0.13	0.02	0.00	-0.00	-0.02
high_prec_dur	0.11	0.02	0.00	-0.00	-0.04
inter_mod_perc	0.13	0.02	0.00	-0.00	-0.05

importance of at least 0.05 for 75% of the basins in the training set. At the second spot we have `baseflow_index_ceh`. This attribute has median and 25th percentile importances similar to those of `Q95`, but is significantly less important for 25% of the basins in the training set.

The top 20 importances found by $US_{lstm,\ all}$ (cross validated) using the permutation algorithm are shown in Table 5.2. The top two attributes are `low_prec_dur` and `low_prec_freq`. These have significantly higher 75th percentile and upward importances than all other attributes. The third most important attribute is `tawc` and has a 75th percentile importance that is roughly half of the above ranked attribute. The highest ranked attribute has an importance of 0.09 for 25% of the basins in the training set. Of these 20 attributes seven (`low_prec_dur`, `low_prec_frec`, `high_prec_frec`, `p_seasonality`, `frac_snow`, `p_mean`, `high_prec_dur`) are climatic indices derived from the time series the model is trained on. The attribute `aridity` is also a climatic indice, but is not based on any time series accessed by the model. Three attributes (`grass_perc`, `crop_perc, inwater_perc`) are land cover attributes. Three attributes (`tawc`, `root_depth`, `conductivity_hypres`) are soil attributes. Four attributes (`elev_90`, `elev_10`, `elev_50`, `area`) are

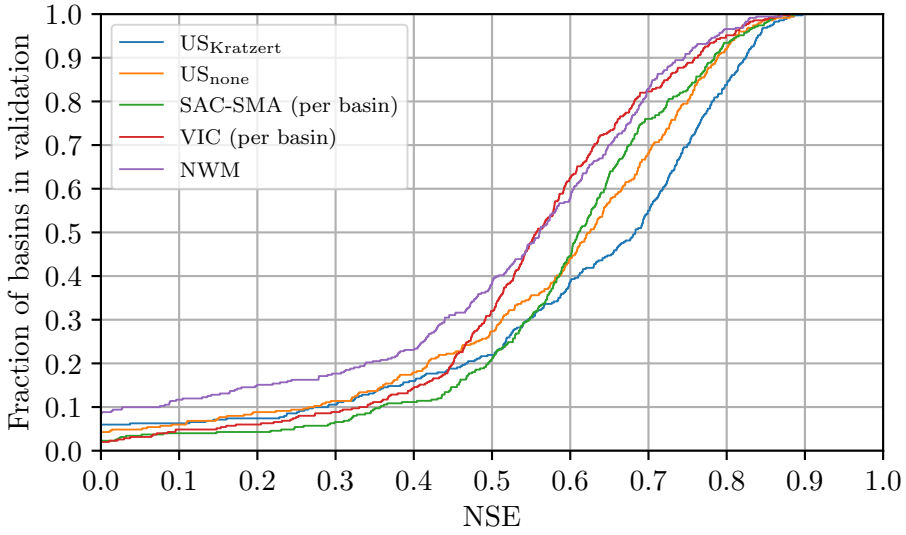

Figure 5.3: Cumulative distribution function of the NSE score of models trained on CAMELS [*Addor et al.*, 2017]. The LSTM models $US_{Kratzert}$ and US_{None} are described in Table 4.1 and are based on the models originally trained by *Kratzert et al.* [2019a]. The SAC-SMA and VIC benchmarks are provided by *Kratzert* [2019a], NWM by *NOAA* [2018]; *Kratzert et al.* [2019a].

topographic. Two attributes (`no_gw_perc` and `inter_mod_perc`) are hydrogeologic attributes. Details of these attributes can be found in Table 2 in *Coxon et al.* [2020] and references therein.

5.2 Models trained on CAMELS

In this section we present the results related to model selection and feature importance when training and predicting on CAMELS [*Addor et al.*, 2017]. This section is meant to show whether we are able to produce results similar to those of *Kratzert et al.* [2019a] in a way that is comparable to the rest of our experiments. In addition, we present apparent feature importance of these trained models.

5.2.1 Performance

Figure 5.3 shows the performance of our reimplementation of the LSTM models created by *Kratzert et al.* [2019a] along with three traditional hydrological models. Table 4.1 contains more details on the LSTM models. "VIC" is the Variable Infiltration Capacity model calibrated on CAMELS. "SAC-

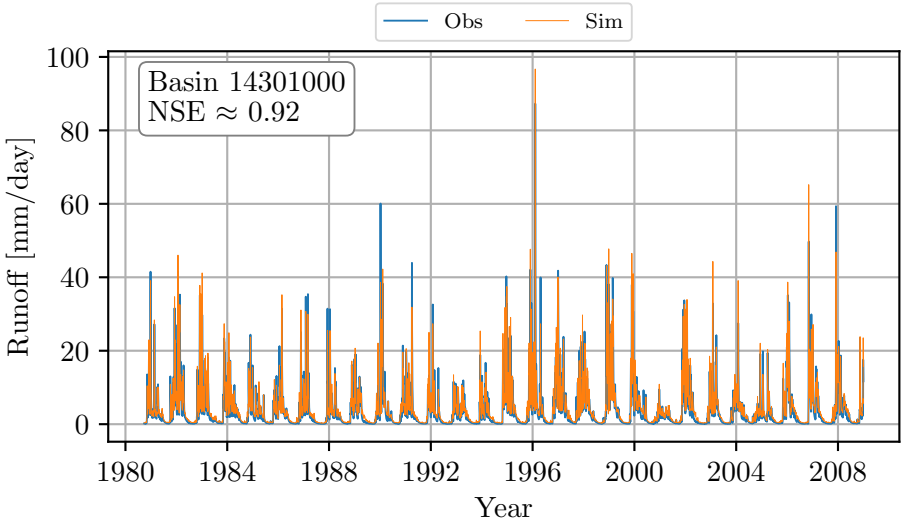

Figure 5.4: Highest scoring prediction on the validation set made using $US_{Kratzert}$ compared to the observed outcome. "Obs" is observed runoff, "Sim" is predicted runoff. The basin data is taken from CAMELS [*Addor et al.*, 2017].

SMA" is the SACramento Soil Moisture Accounting model calibrated on CAMELS. These benchmarks are provided by *Kratzert* [2019a] and are originally created by *Newman et al.* [01 Aug. 2017], and are trained per-basin as opposed to the other models. "NWM" is a benchmark of the National Water Model run on CAMELS. This benchmark is available without any licensing at *NOAA* [2018] and we use an already preprocessed version provided by *Kratzert et al.* [2019a]. The results here indicate that the process-driven models VIC and NWM perform similarly, SAC-SMA and US_{none} perform similarly and better than the process-driven models and that there is a clear performance gap in favour of $US_{Kratzert}$. The LSTM models ($US_{Kratzert}$ and US_{none}) both perform with NSE values below zero for approximately five percent of the basins during cross validation.

The highest scoring validation set prediction made using $US_{Kratzert}$ is shown in Figure 5.4. It has an NSE score of ≈ 0.92. As in the previous section we observe that the peak discharges are often either overestimated or underestimated.

5.2.2 Importance

The permutation importances of the static basin attributes used by $US_{Kratzert}$ ranked by median importance are shown in Table 5.3. Three out of the four

Table 5.3: Permutation importance of all static basin attributes used by $US_{Kratzert}$ (same model configuration as in *Kratzert et al.* [2019a] refit to the cross validation split used in this thesis), ranked by median importance.

	95%	75%	Median	25%	5%
area_gages2	0.42	0.09	0.02	0.00	-0.02
frac_snow	0.26	0.06	0.02	0.00	-0.02
aridity	0.33	0.10	0.02	0.00	-0.03
elev_mean	0.25	0.06	0.02	0.00	-0.02
gvf_diff	0.13	0.04	0.01	-0.00	-0.04
low_prec_freq	0.15	0.04	0.01	-0.00	-0.03
geol_permeability	0.12	0.04	0.01	-0.00	-0.05
slope_mean	0.19	0.04	0.01	-0.00	-0.03
clay_frac	0.17	0.05	0.01	-0.00	-0.04
high_prec_freq	0.15	0.04	0.01	-0.00	-0.03
soil_porosity	0.12	0.04	0.01	-0.00	-0.03
p_mean	0.23	0.05	0.01	-0.00	-0.04
sand_frac	0.13	0.04	0.01	-0.00	-0.04
p_seasonality	0.11	0.03	0.01	-0.00	-0.03
low_prec_dur	0.17	0.04	0.01	-0.00	-0.02
frac_forest	0.14	0.03	0.01	-0.00	-0.03
gvf_max	0.16	0.03	0.01	-0.00	-0.03
pet_mean	0.11	0.03	0.01	-0.00	-0.06
lai_diff	0.12	0.03	0.01	-0.00	-0.03
silt_frac	0.14	0.03	0.01	-0.01	-0.04
lai_max	0.11	0.02	0.01	-0.00	-0.03
max_water_content	0.09	0.02	0.01	-0.00	-0.02
high_prec_dur	0.08	0.02	0.00	-0.00	-0.03
soil_depth_statsgo	0.08	0.02	0.00	-0.00	-0.03
soil_depth_pelletier	0.15	0.04	0.00	-0.00	-0.03
soil_conductivity	0.09	0.01	0.00	-0.00	-0.02
carbonate_rocks_frac	0.12	0.01	0.00	-0.00	-0.02

top ranked attributes are also ranked in the top four by *Kratzert et al.* [2019b] using the same attribute subset, but a different ranking method.

5.3 Models trained on CAMELS and CAMELS-GB

In this section we present results related to model selection and feature importance when training and validating using both CAMELS and CAMELS-GB as a combined dataset. This section is meant to show whether we can obtain satisfactory performance on ungauged basins from both datasets with the same model.

5.3.1 Performance

Figure 5.5 shows the cross validated performance of $\text{Mixed}_{\text{lstm}}$, $\text{Mixed}_{\text{ea-lstm}}$ and $\text{Mixed}_{\text{none}}$ trained on a dataset consisting of both CAMELS and CAMELS-GB. See Table 4.1 for information on these LSTM models. On CAMELS-GB the models with basin attributes perform significantly better than the model trained without them. This also applies to the performance on CAMELS, but here the difference is smaller. The EA-LSTM and LSTM models with basin attributes perform similarly, having a median NSE of 0.77 on CAMELS-GB and 0.65 on CAMELS. Compared to the best performing models in Figure 5.1 and 5.3 the performance of the mixed model is comparable, although lower.

Figure 5.6 shows the best predictions made by $\text{Mixed}_{\text{lstm}}$ on CAMELS-GB and CAMELS. The best performing basin by $\text{US}_{\text{Kratzert}}$ is the same as for $\text{Mixed}_{\text{lstm}}$. For CAMELS-GB we also include a prediction made on basin 50001 as that is the best performing basin for $\text{GB}_{\text{lstm, all}}$. For CAMELS the best prediction of the combined model performs with an NSE 0.01 higher than that of the model trained on CAMELS. For CAMELS-GB it performs 0.02 lower.

5.3.2 Importance

Table 5.4 shows the attributes used by $\text{Mixed}_{\text{lstm}}$ ranked by median importance as found using the permutation algorithm. Sorted by median importance on basins from CAMELS-GB. The highest median importance attribute on basins from CAMELS is `pet_mean`, the highest on CAMELS-GB is `aridity`.

As a summary of our importance analysis we include the performances of $\text{GB}_{\text{lstm, all}}$, $\text{US}_{\text{Kratzert}}$, SAC-SMA trained basin-wise, VIC trained basin-wise, NWM and $\text{Mixed}_{\text{lstm}}$ plotted against `runoff-ratio` (An attribute not used by any of the models. It is the full time series averaged ratio between precipitation and runoff.) in an effort to analyze whether our LSTM models

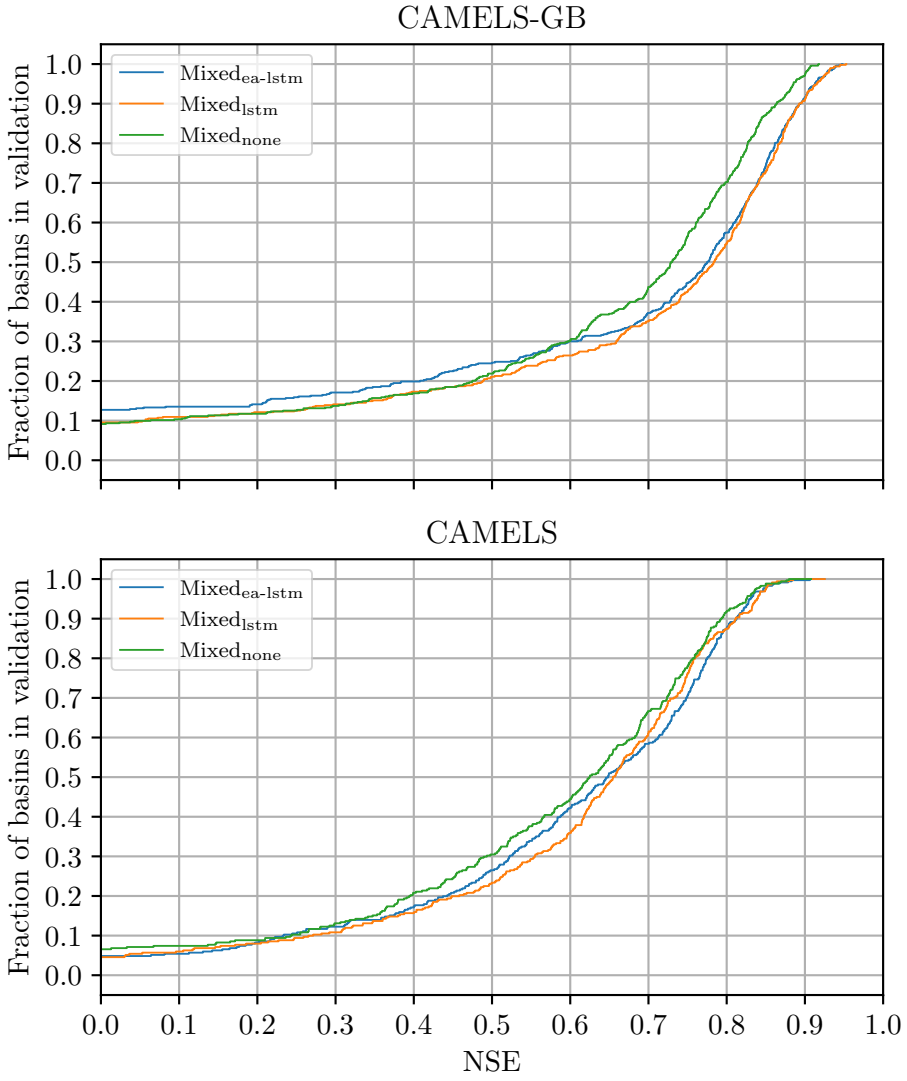

Figure 5.5: Cumulative distribution function of the NSE score of LSTM models trained on a dataset consisting of both CAMELS *Addor et al.* [2017] and CAMELS-GB [*Coxon et al.*, 2020]. The models are described in Table 4.1. The top figure shows the performance of the models on CAMELS-GB and the bottom figure shows the performance on CAMELS. The top figure is the cross validated performance on CAMELS-GB, the bottom figure on CAMELS.

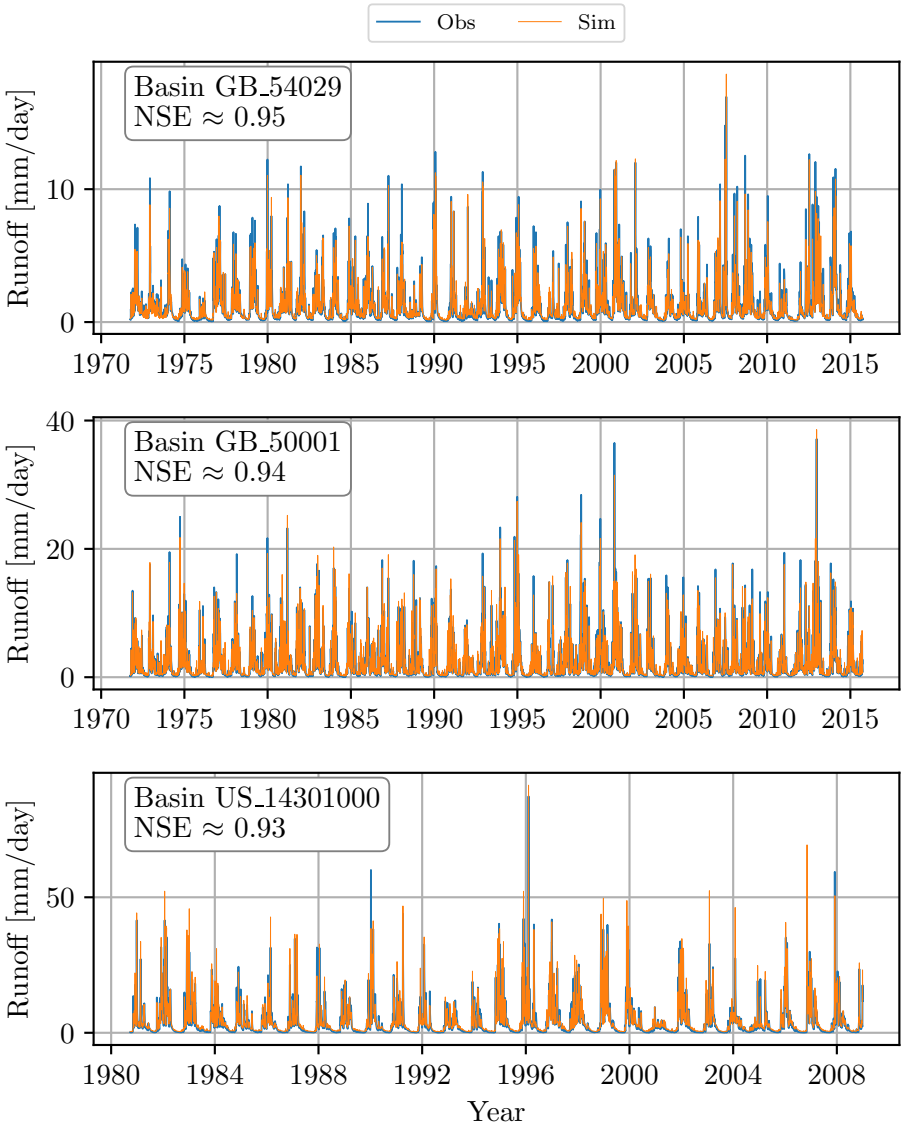

Figure 5.6: Best predictions by $\text{Mixed}_{\text{lstm}}$ on CAMELS-GB (top) [*Coxon et al.*, 2020] and CAMELS (bot) [*Addor et al.*, 2017] including catchment IDs and NSE.

Table 5.4: Attribute importance ranked by median importance on CAMELS-GB [*Coxon et al.*, 2020] as found using the permutation algorithm on $\text{Mixed}_{\text{lstm}}$. Each percentile column is split into two subcolumns, one for CAMELS [*Addor et al.*, 2017] and one for CAMELS-GB.

	95%		75%		Median		25%		5%	
	GB	US	GB	US	GB	US	GB	US	GB	US
aridity	0.66	0.35	0.22	0.06	0.06	0.01	0.01	-0.00	-0.04	-0.05
high_prec_freq	0.43	0.22	0.14	0.07	0.04	0.02	0.00	-0.00	-0.05	-0.06
low_prec_freq	0.73	0.14	0.14	0.05	0.04	0.01	0.01	-0.00	-0.04	-0.05
p_seasonality	0.50	0.33	0.15	0.08	0.03	0.02	0.00	-0.00	-0.04	-0.05
area	0.38	0.39	0.10	0.10	0.03	0.02	0.00	-0.00	-0.04	-0.06
gvf_max	0.48	0.22	0.11	0.06	0.03	0.02	0.00	-0.00	-0.04	-0.04
pet_mean	0.84	0.30	0.12	0.13	0.02	0.05	0.00	0.01	-0.02	-0.06
clay_perc	0.35	0.18	0.09	0.07	0.02	0.02	-0.00	-0.00	-0.07	-0.05
silt_perc	0.42	0.27	0.08	0.07	0.02	0.01	-0.00	-0.01	-0.06	-0.07
p_mean	0.36	0.19	0.06	0.04	0.01	0.01	-0.00	-0.00	-0.07	-0.05
frac_forest	0.41	0.36	0.08	0.12	0.01	0.04	-0.00	0.00	-0.04	-0.04
sand_perc	0.31	0.21	0.07	0.06	0.01	0.01	-0.00	-0.00	-0.06	-0.06
high_prec_dur	0.31	0.19	0.06	0.05	0.01	0.01	0.00	-0.00	-0.04	-0.04
porosity_cosby	0.22	0.15	0.06	0.04	0.01	0.01	-0.00	-0.00	-0.07	-0.05
low_prec_dur	1.28	0.27	0.06	0.04	0.01	0.01	-0.00	-0.00	-0.04	-0.04
conductivity_cosby	0.55	0.11	0.09	0.03	0.01	0.00	-0.00	-0.00	-0.05	-0.05
organic_perc	0.21	0.10	0.04	0.01	0.00	0.00	-0.01	-0.00	-0.06	-0.02
frac_snow	0.32	0.41	0.06	0.10	0.00	0.02	-0.00	0.00	-0.03	-0.03
soil_depth_pelletier	0.28	0.25	0.02	0.03	0.00	0.00	-0.00	-0.00	-0.05	-0.02

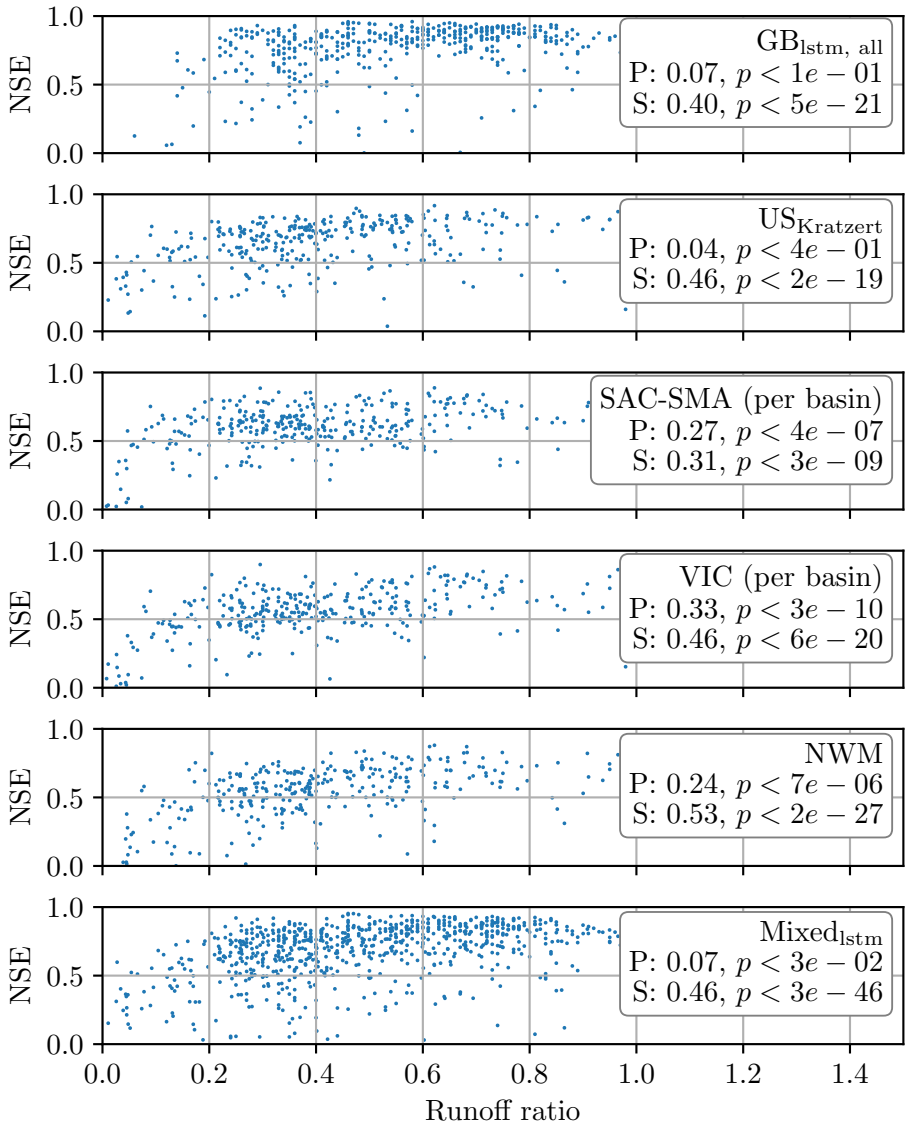

Figure 5.7: Rainfall-runoff ratio scatter plotted against NSE values per basin. Pearson (P) and Spearman (S) correlation coefficients as well as their respective p-values are shown in the figure.

interact differently with low rainfall-runoff ratio basins. Figure 5.7 shows that all models have Spearman correlations $\in (0.33, 0.53)$ with significant p-values. Only the traditional models show linear correlation, implying a less direct relationship between the rainfall-runoff ratio and expected performance for LSTM models.

5.4 Models trained for transfer learning

In this section we present results related to model selection and feature importance when training on CAMELS-GB and validating on CAMELS (GB→US) and vice-versa. This section is meant to show whether we can use information in one dataset to be able to satisfyingly make predictions on the other.

5.4.1 Performance

Figure 5.8 shows the cross validated performance of all models trained on CAMELS and validated on CAMELS-GB and vice-versa. These models are denoted as $\text{Transfer}_{\text{GB/US, model-type}}$ For more information on the models, see Table 4.1. No models trained on CAMELS-GB perform at a satisfactory level on basins from CAMELS. The models trained on CAMELS all perform at a lower level on CAMELS-GB than models that are trained on CAMELS-GB. The best performing transfer model is $\text{Transfer}_{\text{US, none}}$, it has a median NSE of 0.52 on CAMELS-GB. Compared to $\text{Transfer}_{\text{GB, lstm}}$, which has a median NSE of 0.77, this performance is significantly lower. The transfer models also perform worse than expected on the dataset they are trained on as the chosen epoch is based on the performance on both datasets. This is illustrated in Figure 5.9, which shows boxplots of the performance per epoch of $\text{Transfer}_{\text{US, none}}$ and $\text{Transfer}_{\text{GB, none}}$ per training epoch on both CAMELS-GB (top) and CAMELS (bottom). We observe that the performance of $\text{Transfer}_{\text{US, none}}$ increases on CAMELS-GB up until epoch 5 and then gradually decreases while it increases steadily and converges on CAMELS around epoch 14. The epoch optimized for both datasets which is shown in Figure 5.8 is epoch 9. For $\text{Transfer}_{\text{GB, none}}$ this is more apparent. Epoch 1 is the best epoch for the performance on CAMELS, while the performance on CAMELS-GB seems to increase for every epoch. The optimized epoch found for this model on both datasets is epoch 24.

The best predictions made by $\text{Transfer}_{\text{US, lstm, none}}$ on CAMELS and CAMELS-GB are shown in Figure 5.10. In addition, predictions on basins 50001 from CAMELS-GB and 14301000 from CAMELS are included in the figure to compare with the performance of $\text{GB}_{\text{lstm, all}}$ shown in Figure 5.2 and $\text{US}_{\text{Kratzert}}$ shown in Figure 5.4. The model performs worse on CAMELS than $\text{US}_{\text{Kratzert}}$, worse on CAMELS and CAMELS-GB than $\text{Mixed}_{\text{lstm}}$ and worse on CAMELS-GB than $\text{GB}_{\text{lstm, all}}$.

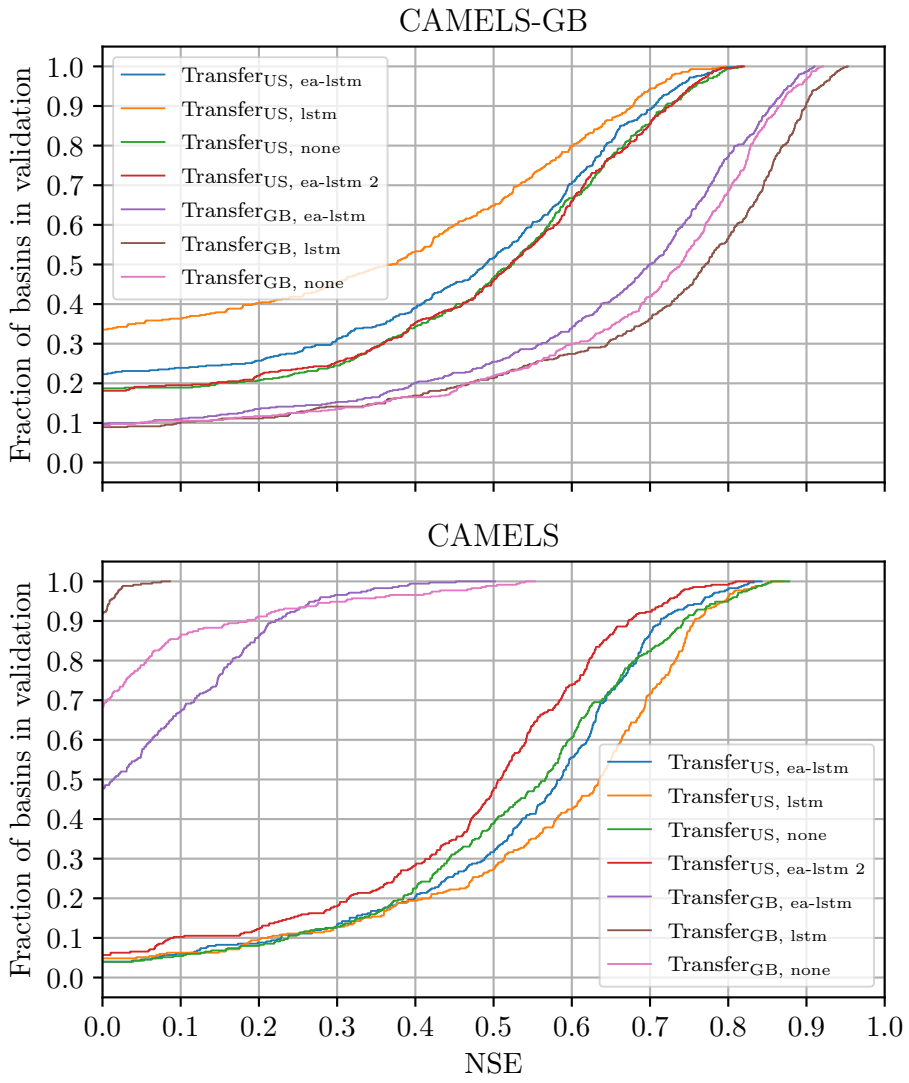

Figure 5.8: Cumulative distribution function of the NSE score of LSTM models trained on CAMELS [*Addor et al.*, 2017] and validated on a validation part of CAMELS as well as the entirety of CAMELS-GB [*Coxon et al.*, 2020]. The top figure shows the performance of the models on CAMELS-GB and the bottom figure shows the performance on CAMELS.

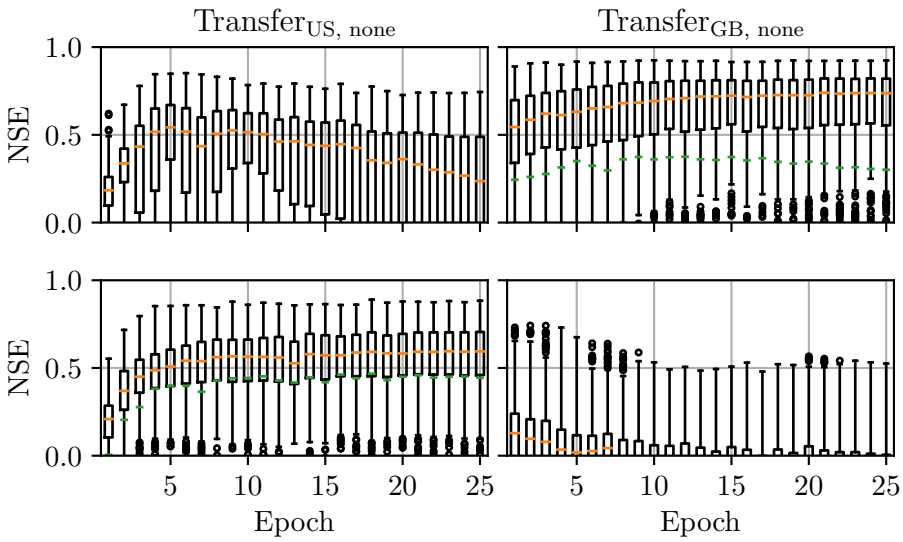

Figure 5.9: Cross validated performance per epoch of Transfer$_{\text{US, none}}$ and Transfer$_{\text{GB, none}}$ on CAMELS-GB (top) and CAMELS (bottom). The orange lines indicate the median NSEs, while the green lines indicate the mean NSEs.

As both cases (GB → US and US → GB) show that the transfer learning works better without static attributes, we do not include an analysis of the importance in the transfer learning case.

5.5 Test performance and summary.

In this section we present test results from the models we choose from Section 5.1-5.4 in this chapter and compare with the cross validation performance. We use the test set performance to get an expected level of real world performance, in contrast to cross validation, which is used to compare models and optimize hyperparameters.

The top plot in Figure 5.11 shows GB$_{\text{lstm, all}}$ refit on the full training set, trained for the optimal amount of epochs found via cross validation and validated on the test set. This is compared to the cross validated performance of the same model. We observe that cross validated and testing performance are similar.

Second from the top in Figure shows US$_{\text{Kratzert}}$ refit on the full training set compared to the cross validated performance. The performance is similar, although notably lower between the 40th and 70th percentile performant basins.

Third from the top we see the test set performance of Mixed$_{\text{lstm}}$ refit

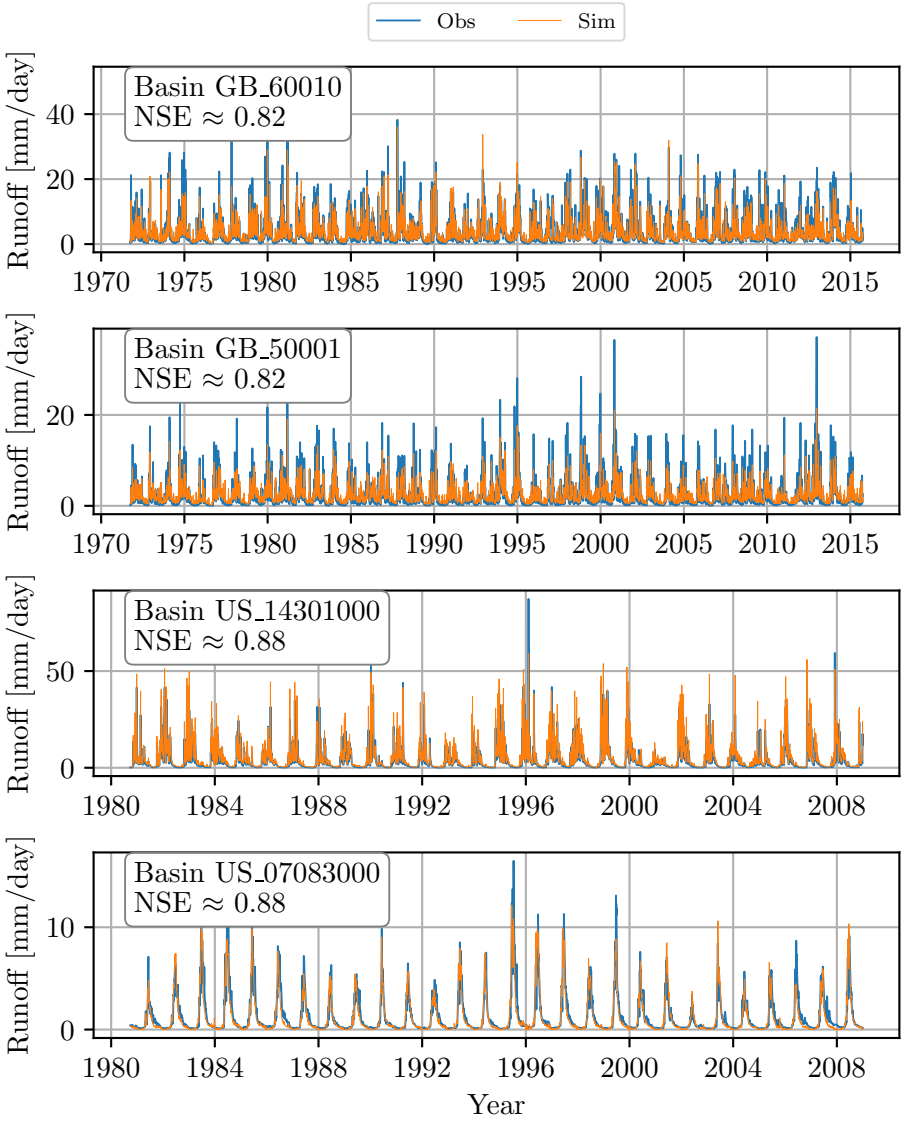

Figure 5.10: Best predictions by Transfer$_{\text{US, lstm, none}}$ on CAMELS-GB (top) [*Coxon et al.*, 2020] and CAMELS (bot) [*Addor et al.*, 2017] including catchment IDs and NSE.

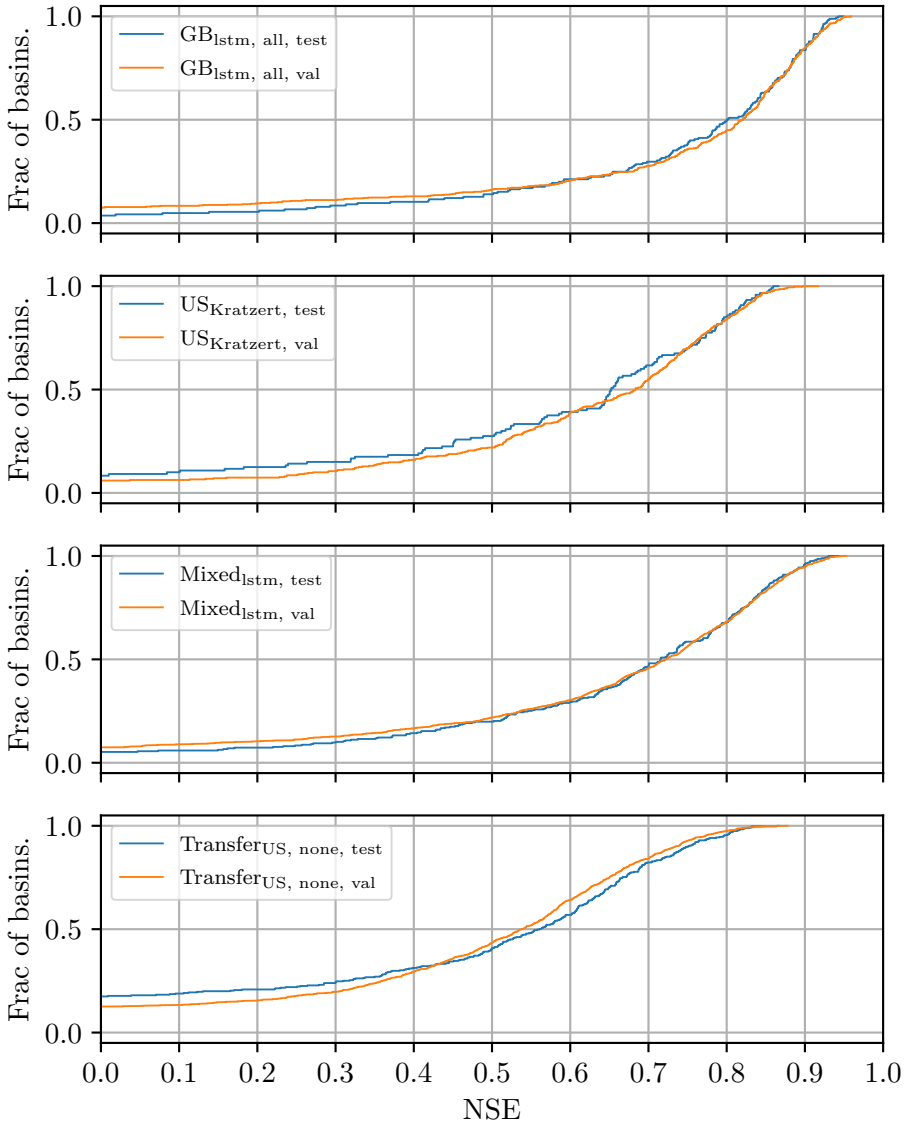

Figure 5.11: From the top: $\text{GB}_{\text{lstm, all}}$, $\text{US}_{\text{Kratzert}}$, $\text{Mixed}_{\text{lstm}}$ and $\text{Transfer}_{\text{US, lstm, none}}$ refit on their respective full train datasets and validated on their test sets compared with the cross validated performance on their train sets.

Table 5.5: Summary table showing median validation and test NSE values of the best models chosen from Section 5.1-5.4 as well as the epoch optimized using cross validation.

Model	Median NSE CV GB	Median NSE CV US	Median NSE Test GB	Median NSE Test US	Epoch
$GB_{lstm,\ all}$	0.82	-	0.80	-	15
$US_{Kratzert}$	-	0.68	-	0.65	13
$Mixed_{lstm}$	0.78	0.66	0.78	0.67	15
$Transfer_{none}$	0.52	0.57	0.54	0.59	9

on the full training set and validated on the test set. The graphs overlap almost perfectly.

The bottom plot is the test set performance of $Transfer_{US,\ lstm,\ none}$ refit on the full CAMELS training set. This is compared to the cross validated performance on CAMELS merged with the ensemble performance on CAMELS-GB. The graphs are similar in shape, although a notably higher amount of basin predictions have a negative NSE value. Finally, the validation and test performances as well as optimized epochs of the best models from Section 5.1-5.4 are shown in Table 5.5. For the models validated and tested on Both CAMELS and CAMELS-GB the median NSE values are shown separately. The optimized epochs are also shown. For more details on the models in question see Table 4.1.

Chapter 6

Discussion

In this chapter we discuss the performance of our LSTM models. We discuss the implications our results and the results of similar works may have on the physical understanding of rainfall-runoff modelling. In addition we discuss potential ways to improve machine learning models for rainfall-runoff modelling, with focus on performance and interpretability.

6.1 Model Selection

The results presented in Section 5.1-5.3 all agree that performance is increased when introducing static attributes. In addition to this, we see that the ordinary LSTM models that treat the static attributes as time series are the highest performing models. We find it likely that this is due to the decrease in model complexity in an EA-LSTM compared to an LSTM. For example, $\text{Mixed}_{\text{ea-lstm}}$ has 205057 trainable parameters (weights and biases), $\text{Mixed}_{\text{lstm}}$ has 285953 and $\text{Mixed}_{\text{none}}$ has 266497. The EA-LSTM is a less complex model than an ordinary LSTM trained without static attributes. The EA-LSTM's performance coming close to that of an LSTM, while only using 72% of the trainable parameters is notable, however. It could imply that even a less complex model benefits from the additional information contained in the attributes. In all cases but the case of transfer learning, our best performing models are the most complex model with the highest amount of static attributes possible. For prediction on CAMELS-GB that means $\text{GB}_{\text{lstm, all}}$, for CAMELS the best model is $\text{US}_{\text{Kratzert}}$, for the datasets mixed the best model is $\text{Mixed}_{\text{lstm}}$, and for transfer learning: $\text{Transfer}_{\text{US, none}}$.

We do not spend significant time in this thesis tuning hyperparameters. Instead, we choose to use the hyperparameters derived by *Kratzert et al.* [2019b]. The one exception to this is for models trained on CAMELS-GB, shown in Section 5.1. We found that using dropout in this case at best makes no difference and at worst slightly decreases performance. In some

cases we also tested whether increasing the sequence length from 270 days to 365 days increases performance, but our results indicate that this leads to no performance difference. To save memory and decrease training time we therefore keep the lower sequence length of 270 days.

We acknowledge that it may be possible to further increase the performance by doing a thorough hyperparameter tuning experiment. The tuning done by *Kratzert et al.* [2019b] is a simple grid search with few points, and is not guaranteed to be a good basis, especially since this tuning was done for gauged basin prediction. Hyperparameters are likely to have different optimal values for our case of ungauged prediction. A very thorough hyperparameter tuning would be hard to brute force, due to the computationally expensive nature of our models, therefore one would have to implement a more sophisticated method than mere exhaustive grid searching. This becomes even more computationally expensive if one wants to tune using cross validation.

We find no clear signs of overfitting in our selected models when comparing cross validated performance to test performance. Table 5.5 indicates similar performance between the cross validated models and the refit models validated on the test set. There are some discrepancies in the CDF plot of the performance of the models trained on CAMELS shown in Figure 5.11 (second from top), leading to a median NSE of 0.65 on the test set as opposed to 0.68 in the cross validated case. This small deviation could perhaps be explained by the findings of *Kratzert et al.* [2019b], which found that an ensemble of LSTM models performs better than a singular LSTM because of nontrivial differences in performance caused by randomness in the training process. For computational reasons we have chosen not to use model ensembles anywhere in this thesis.

6.2 Performance and Importance Analysis

6.2.1 CAMELS-GB

Table 5.1 shows among others `Q95` as the most important attribute for the overfit "proof of concept" model. This attribute is the 95th percentile runoff value for the full runoff time series for a given basin, which is information directly derived from the expected outcome. Figure 5.1 also shows that the overfit model drastically outperforms all other models. Because of this, we believe it is safe to assume that an LSTM model can learn which static attributes are the most important in CAMELS-GB, at least in the most obvious case. In addition, it also indicated that the permutation feature importance algorithm succeeds at deciding the most important static attribute in this case. These assumptions are therefore used to strengthen our confidence in the results from our other, non-overfit models.

Looking again at Figure 5.1 along with Table 5.5, we see that the median

NSE value of $GB_{lstm,\ all}$ is approximately 0.82 cross validated, with a test score of 0.80. This performance is vastly superior to that found by *Kratzert et al.* [2019a], and is even better than gauged prediction on CAMELS, such as *Kratzert et al.* [2019b]. This makes sense when one considers the relatively low variance in climate in Great Britain, compared to the United States. CAMELS-GB consists of the same amount of basins as CAMELS, but with less variance this means the basins in the validation sets will be more similar to the ones in the training set, making it less critical for the model to generalize. As far as the author knows, there are no published benchmarks using traditional models on CAMELS-GB. Creating such a benchmark in the future is necessary for us to properly assess model performance.

In Figure 5.2, we see that at least for the best performing prediction, the high NSE score does in fact transfer to a visually impressive performance. Despite some peaks not being correctly simulated, the LSTM model manages to replicate the observed runoff in a very acceptable manner.

As the results presented in Table 5.2 indicate that the many of the important static attributes in CAMELS-GB (as perceived by $GB_{lstm,\ all}$) are derived directly from the input time series (the most important attribute being `low_prec_dur`, for instance), it is likely that we cannot extract much new information from these, as this information should be available in the time series already used by the models. If this were the case for CAMELS-GB only, one could perhaps assume it being caused by CAMELS-GB having more similar basins, but later we see that this is also the case for models predicting on CAMELS. The reason for this being the case is likely connected to the nature of the LSTM training process and the sequence length. Ways to further improve the long-term memory of an LSTM should be further explored in the future. The biggest examples of this type of attribute are the top two ranked attributes: `low_prec_freq` and `low_prec_dur`. These severely outrank all other static attributes, especially for the 75th percentile of importances, and they are directly derived from the precipitation time series. They are defined as the frequency of days with ≤ 1 mm/day precipitations, and the average duration of these dry periods respectively. Other than derived features, the model ranks the other attributes relatively similarly for the upper 75th percentile. These other attributes mostly contain information about land cover and elevation levels, which is information an LSTM cannot access through the time series contained in CAMELS-GB. Process-driven models such as VIC use information such as land coverage and elevation by splitting the catchment into a spatial grid, it is therefore unlikely that any of these attributes can contribute to any major improvements for process-driven models for predicting on CAMELS-GB. To summarize: Our model tends to mostly prefer static attributes containing information already available in the time series, it is therefore hard to argue that the static attribute ranking found here can be used to improve traditional models, which already use this information via the time series.

6.2.2 CAMELS

Looking at Figure 5.3 and Table 5.5, the performance of $US_{Kratzert}$ coincides with the results of *Kratzert et al.* [2019a], which is what we wanted to replicate. The performance is generally lower than that of $GB_{lstm,\ all}$, which makes sense when one considers how much more diverse the United States climate is, compared to the climate of Great Britain.

The best prediction made by $US_{Kratzert}$ is very accurate and has an NSE of 0.92. This is lower than for the best prediction on CAMELS-GB. Similarly to the previous section, we here too see that the prediction fails to predict the higher peaks of runoff. This is also something observed by *Kratzert et al.* [2018].

Table 5.3 shows that the permutation algorithm yields different results than the feature ranking done by *Kratzert et al.* [2019b]. Said paper averaged the normalized sensitivity (derived by what they call the explicit Morris method) of each basin, while we present percentiles and rank based on the median [1]. Looking at our median scores, we see that the majority of basins do not benefit from the inclusion of static attributes. Looking at the upper percentiles, the most important attributes in our case still somewhat agree with those of *Kratzert et al.* [2019b]. Topological information and climatic indices are important. By far the most important attributes for the 75th percentile are aridity and catchment area. Our results do however favour land coverage types and soil types more than the results of *Kratzert et al.* [2019b], and the attribute deemed the most important by said paper, `p_mean`, is only significantly important for the 95th percentile of basins. As expected, `frac_snow` is much more important for CAMELS than for CAMELS-GB, as there is more snow in certain regions in the United States than in Great Britain. This is also reflected in the fact that `frac_snow` is important for a small portion of British basins (the 95th percentile), while it is more important both for the 75th and 95th percentile in the US.

6.2.3 Mixed model

Our best model trained on both CAMELS and CAMELS-GB seems to find static attributes more important than the models trained on the datasets individually. Table 5.4 shows that the model also ranks the attributes differently. Figure 5.5 shows that the addition of static attributes is of significantly higher benefit for prediction on CAMELS-GB than for CAMELS although the performance per dataset is not very far off the individually trained models (see Figure 5.5). This could be consistent with the importance rankings, as the median importances, and especially the 75th percentile and above importances, are significantly higher on CAMELS-GB

[1] Also note that we predict on ungauged basins, in contrast to said paper. This could differentiate important attributes.

than those on CAMELS. There is a chance that this is due to the model learning to use the attributes to differentiate between the two datasets, without actually extracting important information. This could also be because CAMELS-GB is more homogeneous, and shuffling the attributes in CAMELS-GB therefore has a lesser effect than shuffling those of CAMELS.

An interesting detail is that the permutation algorithm yields negative importance for some basins. This happens for all models mentioned in this section. This implies that our LSTM models actually perform worse on some basins because of the inclusion of static attributes, perhaps due to a lack of training data representing similar basins in the dataset. This is likely alleviated by introducing more data. We believe other ways to rank attributes should also be attempted in the future. *Kratzert et al.* [2019b] used a robustness test which involves gradually adding Gaussian noise to a feature and evaluating how it affects model performance. *Addor et al.* [2018] uses a different method altogether, the results of which *Kratzert et al.* [2019b] agree with. To our knowledge there exists no feature ranking results on CAMELS-GB apart from what is shown in this thesis, so these two other methods should also be attempted on CAMELS-GB in addition to the permutation feature importance algorithm employed in this thesis to get further context and comparability.

6.2.4 Transfer learning

The results in Section 5.4 show that the models trained for transfer learning do not benefit from adding static attributes. Therefore, we do not include a permutation algorithm for this case, We also see that the case of US→GB significantly improves when removing attributes. This makes sense as Great Britain has much smaller climatic variance than the United States (see for instance Figure 4.2). We believe these two phenomena could be related. It is likely that the relatively lower performance from using static attributes comes from the fact that the datasets have different climatic conditions. There could also be room for improvement in how we preprocess both the attributes and the time series. As seen in Table 4.3 we assume that the temperature time series of CAMELS behave like symmetrical periodic functions, therefore leading to

$$T_{\text{average}} \approx \frac{T_{\text{min}} + T_{\text{max}}}{2}. \tag{6.1}$$

Ideally, we need to have temperature as a time series for a given model to be able to properly model snow and frozen soil accumulation, which is very important for hydrologic models. The fact that models trained on both datasets still perform comparably to models trained on only one dataset seems to suggest that (6.1) is not entirely destructive, however. In the later stages of this thesis it was discovered that the average temperature is actually present in CAMELS. The older forcing data [*Maurer et al.*, 2002]

contains this information stored as duplicates in the maximum and minimum temperature time series. Due to time constraints, we are unable to train models utilizing this. This is likely a good start for those looking to improve transfer learning performance. The best outcome is likely to come from finding maximum and minimum temperatures for CAMELS-GB, however.

It seems to us that there are several angles to further tackle the issue of transfer learning between these datasets. One of which we stated above: Find maximum and minimum time series for CAMELS-GB. A second way is to improve the preprocessing of static attributes to the point where one would hopefully see better performance with attributes than without.

While we stated earlier in this chapter that we see no clear signs of overfitting, this is not the case for transfer learning. Figure 5.9 indicates that $\text{Transfer}_{\text{GB, none}}$ and $\text{Transfer}_{\text{US, none}}$ start overfitting on their respective training datasets at epoch one and five respectively. This overfitting is not apparent when validating on basins from the same dataset as they are trained on, but very quickly manifests when validating on the other dataset. While improved pre-processing, regularization and the gathering of more cross-compatible data could alleviate this problem, it is likely that this overfitting is unavoidable using the current datasets. More data, both in quantity and in diversity, is likely needed for the model's training process to be able to successfully predict on a dataset not included during training.

There now exist several other similar datasets to the two used for this study. The easiest way to improve any machine learning model is often to provide more data for training. A Chilean dataset consisting of 515 catchments by the name CAMELS-CL [*Alvarez-Garreton et al.*, 2018] was studied early in this thesis work, but the quality of the data was deemed worse than that of CAMELS and CAMELS-GB. With some initial preprocessing it could be of interest to see whether it is feasible to use this dataset. In addition we know of two other datasets. One goes by the name CAMELS-BR, and is a Brazilian dataset consisting of 897 catchments [*Chagas et al.*, 2020]. The other is CAMELS-AU, and is an Australian dataset consisting of 222 catchments [*Fowler et al.*, 2021]. Especially CAMELS-BR seems to be of interest due to the amount of catchments. Combining datasets is a non-trivial, time consuming task, which we expect will become increasingly hard the more datasets one tries to combine. It is not unlikely that one has to discard almost all static attributes to make the datasets compatible. We therefore expect that some effort to gather new data would also be necessary. It is of course also of interest to see how well machine learning models perform on one or more of these datasets in isolation. We see this as a possible candidate for the start of a new master thesis.

6.3 Comparison to traditional models

We are able to reproduce the results of *Kratzert et al.* [2019b, a]. Our models trained on CAMELS perform significantly better than traditional models. In this thesis we compare to two process-driven models, VIC and NWM, and one conceptual model, SAC-SMA.

What we additionally show in this thesis is that models trained on both CAMELS and CAMELS-GB also perform better on CAMELS than the addressed traditional models. This could be because the models are complex enough to recognize which dataset a given basin is from based on underlying signatures in either the time series or the static attributes. Perhaps this differentiation could stem from differences in data collection practice.

One of many reasons why LSTM models perform better than traditional models on CAMELS in general could be that the LSTM models are better at modelling basins with special conditions leading to a rainfall-runoff ratio different from 1. When the rainfall-runoff ratio is lower than one it usually implies that the rainfall is either stored as snow or frozen soil, that it is vaporized through photosynthesis in local vegetation, or that there is some difficult to model complexity to the subsurface flow in the area. These processes are difficult to model compared to runoff directly caused by rainfall [*Paniconi and Putti*, 2015]. Our results may indicate that this could be the case. Figure 5.7 shows that only the traditional models have a statistically significant linear relationship between performance and rainfall-runoff ratio. The relationships are generally weak, with a Pearson correlation between 0.24 and 0.33, but they are more apparent than for the LSTM models, where there exists no such linear relationship. The impact of this discovery is quite uncertain, however, as there is significant Spearman correlation (nonlinear correlation) for all the models, ranging from 0.31 to 0.53.

The LSTM models are, as briefly mentioned earlier, much less complex spatially. While process-driven models need to grid the spatial dimension to properly model the physical system, an LSTM model finds relationships between input and output data in a one-dimensional manner. This could make LSTMs and other machine learning methods valuable models to use when there is not enough data available to properly set up a traditional model such as the NWM.

Chapter 7

Outlook

Aside from dropout in the final layer, we employ no regularization methods on our models. *Kratzert et al.* [2019a] argue that there is evidence that adding so-called "physical constraints" to models used on CAMELS could be helpful. This is argued because the LSTM models underperform compared to SAC-SMA on some basins. Here the paper compares SAC-SMA to an LSTM trained on all basins, training and validation split time series-wise and not basin-wise. SAC-SMA is a conceptual model and is therefore not usually seen as a purely "physical" model, but it is based more on physical arguments than a generalized machine learning method, as it is "conceptually" based on the real world. Still, both SAC-SMA and the LSTM are essentially parameter based models, without clear physical interpretations of most parameters. Because of this, we are unsure whether the argument that this is evidence for physical constraints is sufficient. More research is needed on the topic. Regularization in the form of physical constraints may yield several benefits, some of which could be [*Karpatne et al.*, 2017]:

- Interpretability: Physical constraints could improve the physical interpretability of the LSTM.
- Improved generalization: The goal of most machine learning regularization is to improve performance by increasing bias and decreasing variance (see Figure 2.1). Increasing bias using physical laws would fit well into the narrative of *Mendoza et al.* [2015] and *Newman et al.* [01 Aug. 2017], although from the opposite angle. These papers argue that process-driven models are often too statistically constrained, due to them being driven almost entirely by interpretable physical processes. *Newman et al.* [01 Aug. 2017] showed that VIC performs at a higher level when tuning more parameters than usual, as these parameters are often pre-set based on physical a-priori knowledge. This is analogous to the bias-variance trade-off in machine learning. An LSTM is much less constrained and therefore performs better, agreeing with the results of the aforementioned papers. Making LSTM

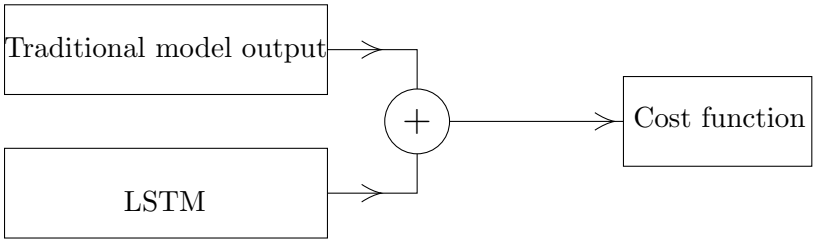

Figure 7.1: Sketch of how a simple hybrid model could be implemented. The traditional model and the LSTM model are completely separate, making it unnecessary to calculate gradients for the traditional model. The idea here would be for the LSTM model to learn the phenomena lacking in traditional models.

models slightly more constrained (as worded by *Mendoza et al.* [2015]) could then make them generalize successfully to more basins while still being powerful enough to learn necessary relationships between input variables.

In the spirit of *Karpatne et al.* [2017]'s argument for increasing the physical consistency of models, we propose a simple semi-physical constraint:

$$\hat{\boldsymbol{y}} = \text{LSTM}(\boldsymbol{x}), \tag{7.1}$$

where $\hat{\boldsymbol{y}}$ now consists of three outputs: $\boldsymbol{y}_{\text{discharge}}$, $\boldsymbol{y}_{\text{frost}}$ and $\boldsymbol{y}_{\text{radiation}}$. These represent runoff, frost/snow and radiated water respectively. Only $\boldsymbol{y}_{\text{discharge}}$ would be treated as the actual input when calculating the original cost function (4.2). We then introduce a long term frost storage variable $\boldsymbol{W}^t_{\text{frost}}$. For each time step this storage variable is updated by

$$\boldsymbol{W}^{t+1}_{\text{frost}} = \boldsymbol{W}^t_{\text{frost}} + \boldsymbol{y}_{\text{frost}}, \tag{7.2}$$

where t is the current time step. When calculating the loss function, we now add a new term to (4.2):

$$L = \text{NSE}^* + \gamma \left| (\boldsymbol{y}_{\text{discharge}} - \boldsymbol{x}_{\text{precipitation}} - \boldsymbol{y}_{\text{radiation}} - (\boldsymbol{y}_{\text{frost}} - \boldsymbol{W}_{\text{frost}})) \right|. \tag{7.3}$$

Here γ is a hyperparameter that needs to be tuned. Our reasoning behind (7.3) is that this cost function penalizes models for making predictions where there is more runoff than available precipitation, snow and frost. Our hypothesis is that this penalty should lead to less complex models that are less likely to predict unphysical behaviors, leading to fewer NSE scores below zero. It could also improve interpretability of the model's output. Whether this or a similar constraint behaves as intended remains to be seen.

Sticking to the topic of increased generalizability and interpretability, we believe it to be of interest to implement a hybrid model, i. e. a machine

learning model implemented alongside or as part of a traditional model. The machine learning model either replaces parts of or supplements the predictions of a traditional model. A very simple implementation of this is shown in Figure 7.1. This model would take the same inputs as the models trained in this thesis, but the cost function would be calculated on the sum of the output of the traditional model and the machine learning model. This means that the machine learning model learns to correct the errors of a given traditional model instead of doing actual prediction.

This thesis and earlier research done by *Kratzert et al.* [2018, 2019b, a] are all limited to simple LSTM models. LSTM models are, due to their recurrent nature, notoriously slow. Graphics cards and similar highly parallel computational devices are not optimized for recurrent calculation. A relatively simple way to surpass this could be to implement a one-dimensional Convolutional Neural Network (CNN) layer as the first layer of our model. This layer could then be used to reduce the time resolution of the data, still retaining information. The usage of CNNs for time series prediction is becoming increasingly widespread [*Zhao et al.*, 2017]. We still argue that LSTM models fit the type of physical system modelled in this thesis very well and is logically consistent with the way rainfall-runoff models are typically structured, but using a CNN layer to improve performance should not detract from this.

Chapter 8

Conclusion

8.1 Summary

We have shown that LSTM models perform well on CAMELS-GB as well as CAMELS, and also that a single model can perform well on both datasets when trained on them at the same time, even with less than optimal preprocessing of the mixed dataset. In all cases models trained and validated on subsets of the same dataset perform approximately equally during cross validation and on the test sets. The models struggle to model the highest peaks and lowest valleys of streamflow, however.

Training on CAMELS and validating on CAMELS-GB and vice-versa yielded disappointing, though not surprising results. We deemed this to be because the datasets cover domains that are too different from one another, as well as lacking equivalent attributes.

Our static attribute analysis yielded no obvious ways to improve the performance of process-driven models, as our models preferred static attributes containing information already used by process-driven models. Better performing machine learning models which are able to extract more information from the time series alone could perhaps alleviate this in the future, by emphasising on more meaningful attributes than those derived directly from the time series.

From the fact (shown first by *Kratzert et al.* [2019a] and recreated in this thesis) that LSTM models outperform traditional models at generalization on CAMELS (0.15 median NSE improvement over the NWM), as well as the fact that we have shown that they perform well on CAMELS-GB, we believe it is possible to claim that machine learning models have the potential to vastly improve the understanding of the physics behind hydrological models. In the future we hope this leads to both improved physical understanding, as well as model quality, in several fields of the physical sciences. Hydrological systems are highly complex, and the need to use large scale data analysis to obtain a better understanding of the underlying physical processes has yet

to be fulfilled.

8.2 Future work

In the previous chapters we mentioned several ways to improve the results of this thesis. Here we give a brief summary.

One of the simplest ways to improve the performance of a machine learning model is to give it more and higher quality training data. To our knowledge there now exists three additional CAMELS-like datasets we have not used in this dataset [*Alvarez-Garreton et al.*, 2018; *Fowler et al.*, 2021; *Chagas et al.*, 2020]. Training LSTM models on these datasets, as well as attempting to combine them with CAMELS and CAMELS-GB is something we believe could improve the generalizability of machine learning models trained for hydrological modelling.

There are likely several ways to improve the performance of our machine learning model through modifying the model architecture. A simple modification to start with could be to implement an LSTM based model with a one-dimensional CNN input layer. This is intended to reduce the number of time steps needed, thus reducing the computational cost of training the model.

Another way to improve the performance of a machine learning model is to improve the hyperparameter tuning. This could be especially important for transfer learning, as the reduction of variance is often important for generalization [*Hastie et al.*, 2009]. This could be a more manageable task if the aforementioned CNN layer is implemented to decrease training time.

A more experimental approach would be to implement a physics based penalty term in the cost function. A simple example of this is described in the previous chapter. This could improve the interpretability of an LSTM and also increase the consistency of the output, as put by *Karpatne et al.* [2017].

A similar, but more advanced approach than stated above, is to employ an LSTM as part of a traditional model in a hybrid approach. This could lead to better generalizability and interpretability. There are several ways to do this, one approach could be to implement an LSTM instead of a given process in a process-driven model. This could then be used to compare to the output of said process in the ordinary model versus the hybrid model. This is likely very computationally expensive, as the process-driven models are more expensive than pure LSTM models. A simpler hybrid model discussed in this thesis is a setup where the LSTM is trained to correct the error of the output of a traditional model. This way one could leverage the already existing model benchmarks on CAMELS [*Kratzert*, 2019a].

Bibliography

Addor, N., A. J. Newman, N. Mizukami, and M. P. Clark, The camels data set: catchment attributes and meteorology for large-sample studies, *Hydrology and Earth System Sciences (HESS)*, *21*(10), 5293–5313, 2017.

Addor, N., G. Nearing, C. Prieto, A. Newman, N. Le Vine, and M. P. Clark, A ranking of hydrological signatures based on their predictability in space, *Water Resources Research*, *54*(11), 8792–8812, 2018.

Alvarez-Garreton, C., et al., The camels-cl dataset: catchment attributes and meteorology for large sample studies-chile dataset, *Hydrology and Earth System Sciences*, *22*(11), 5817–5846, 2018.

Anderson, E. A., *National Weather Service river forecast system: Snow accumulation and ablation model*, vol. 17, US Department of Commerce, National Oceanic and Atmospheric Administration ..., 1973.

Bengio, Y., P. Simard, and P. Frasconi, Learning long-term dependencies with gradient descent is difficult, *IEEE transactions on neural networks*, *5*(2), 157–166, 1994.

Breiman, L., Random forests, *Machine learning*, *45*(1), 5–32, 2001.

Burnash, R. J., R. L. Ferral, and R. A. McGuire, *A generalized streamflow simulation system: Conceptual modeling for digital computers*, US Department of Commerce, National Weather Service, and State of California ..., 1973.

Chagas, V. B., P. L. Chaffe, N. Addor, F. M. Fan, A. S. Fleischmann, R. C. Paiva, and V. A. Siqueira, Camels-br: hydrometeorological time series and landscape attributes for 897 catchments in brazil, *Earth System Science Data*, *12*(3), 2075–2096, 2020.

Cherkauer, K. A., and D. P. Lettenmaier, Hydrologic effects of frozen soils in the upper mississippi river basin, *Journal of Geophysical Research: Atmospheres*, *104*(D16), 19,599–19,610, 1999.

Coxon, G., et al., Camels-gb: hydrometeorological time series and landscape attributes for 671 catchments in great britain, *Earth System Science Data*, *12*(4), 2459–2483, 2020.

Fowler, K. J., S. C. Acharya, N. Addor, C. Chou, and M. C. Peel, Camels-aus: Hydrometeorological time series and landscape attributes for 222 catchments in australia, *Earth System Science Data Discussions*, pp. 1–30, 2021.

Georgakakos, K. P., A generalized stochastic hydrometeorological model for flood and flash-flood forecasting: 1. formulation, *Water Resources Research*, *22*(13), 2083–2095, 1986.

Géron, A., *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*, O'Reilly Media, 2019.

Gers, F. A., J. Schmidhuber, and F. Cummins, Learning to forget: Continual prediction with lstm, 1999.

Gochis, D., et al., Wrf-hydro® v5.1.1, doi:10.5281/zenodo.3625238, 2020.

Gochis, D. J., and F. Chen, Hydrological enhancements to the community noah land surface model, *Tech. rep.*, University Corporation for Atmospheric Research, 2003.

Graves, A., Long short-term memory, in *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, Springer, 2012.

Hamman, J. J., B. Nijssen, T. J. Bohn, D. R. Gergel, and Y. Mao, The variable infiltration capacity model version 5 (vic-5): infrastructure improvements for new applications and reproducibility, *Geoscientific Model Development*, *11*(8), 3481–3496, doi:10.5194/gmd-11-3481-2018, 2018.

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

Hiederer, R., Mapping soil properties for europe—spatial representation of soil database attributes, *Luxembourg: Publications Office of the European Union*, 2013a.

Hiederer, R., Mapping soil typologies-spatial decision support applied to european soil database, *EUR25932EN Scientific and Technical Research Series*, pp. 1831–9424, 2013b.

Hjorth-Jensen, M., Week 40: From stochastic gradient descent to neural networks, https://compphysics.github.io/MachineLearning/doc/pub/week40/html/week40.html, accessed: 2021-07-01, 2020.

Hochreiter, S., and J. Schmidhuber, Long short-term memory, *Neural computation*, *9*(8), 1735–1780, 1997.

Karpatne, A., G. Atluri, J. H. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova, and V. Kumar, Theory-guided data science: A new paradigm for scientific discovery from data, *IEEE Transactions on Knowledge and Data Engineering*, *29*(10), 2318–2331, 2017.

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

Koren, V., M. Smith, and Z. Cui, Physically-based modifications to the sacramento soil moisture accounting model. part a: Modeling the effects of frozen ground on the runoff generation process, *Journal of Hydrology*, *519*, 3475–3491, 2014.

Kratzert, F., Camels benchmark models, hydroshare, doi:https://doi.org/10.4211/hs.474ecc37e7db45baa425cdb4fc1b61e1, 2019a.

Kratzert, F., Camels extended maurer forcing data, hydroshare, doi:https://doi.org/10.4211/hs.17c896843cf940339c3c3496d0c1c077, 2019b.

Kratzert, F., 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*(11), 6005–6022, 2018.

Kratzert, F., D. Klotz, M. Herrnegger, A. K. Sampson, S. Hochreiter, and G. S. Nearing, Toward improved predictions in ungauged basins: Exploiting the power of machine learning, *Water Resources Research*, *55*(12), 11,344–11,354, 2019a.

Kratzert, F., D. Klotz, G. Shalev, G. Klambauer, S. Hochreiter, and G. Nearing, Towards learning universal, regional, and local hydrological behaviors via machine learning applied to large-sample datasets, *Hydrology and Earth System Sciences*, *23*(12), 5089–5110, 2019b.

Leon, J., How do I draw an LSTM cell in Tikz?, TeX Stack Exchange, uRL:https://tex.stackexchange.com/questions/432312/how-do-i-draw-an-lstm-cell-in-tikz/432344 (version: 2021-01-12), 2018.

Liang, X., D. P. Lettenmaier, E. F. Wood, and S. J. Burges, A simple hydrologically based model of land surface water and energy fluxes for general circulation models, *Journal of Geophysical Research: Atmospheres*, *99*(D7), 14,415–14,428, 1994.

Liang, X., E. F. Wood, and D. P. Lettenmaier, Surface soil moisture parameterization of the vic-2l model: Evaluation and modification, *Global and Planetary Change*, *13*(1-4), 195–206, 1996.

Maurer, E. P., A. W. Wood, J. C. Adam, D. P. Lettenmaier, and B. Nijssen, A long-term hydrologically based dataset of land surface fluxes and states for the conterminous united states, *Journal of climate*, *15*(22), 3237–3251, 2002.

Mendoza, P. A., M. P. Clark, M. Barlage, B. Rajagopalan, L. Samaniego, G. Abramowitz, and H. Gupta, Are we unnecessarily constraining the agility of complex process-based models?, *Water Resources Research*, *51*(1), 716–728, 2015.

Miller, D. A., and R. A. White, A conterminous united states multilayer soil characteristics dataset for regional climate and hydrology modeling, *Earth interactions*, *2*(2), 1–26, 1998.

Morris, D., R. Flavin, and R. Moore, *A digital terrain model for hydrology*, 1990.

Nash, J., and J. Sutcliffe, River flow forecasting through conceptual models part i — a discussion of principles, *Journal of Hydrology*, *10*(3), 282 – 290, doi:https://doi.org/10.1016/0022-1694(70)90255-6, 1970.

Newman, A., et al., Development of a large-sample watershed-scale hydrometeorological data set for the contiguous usa: data set characteristics and assessment of regional variability in hydrologic model performance, *Hydrology and Earth System Sciences*, *19*(1), 209–223, 2015.

Newman, A. J., N. Mizukami, M. P. Clark, A. W. Wood, B. Nijssen, and G. Nearing, Benchmarking of a physically based hydrologic model, *Journal of Hydrometeorology*, *18*(8), 2215 – 2225, doi:10.1175/JHM-D-16-0284.1, 01 Aug. 2017.

NOAA, National water model reanalysis, aws, accessed: 2021-07-01, 2018.

Paniconi, C., and M. Putti, Physically based modeling in catchment hydrology at 50: Survey and outlook, *Water Resources Research*, *51*(9), 7090–7129, 2015.

Paszke, A., et al., Pytorch: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, pp. 8024–8035, Curran Associates, Inc., 2019.

Pelletier, J. D., P. D. Broxton, P. Hazenberg, X. Zeng, P. A. Troch, G.-Y. Niu, Z. Williams, M. A. Brunke, and D. Gochis, A gridded global data set

of soil, intact regolith, and sedimentary deposit thicknesses for regional and global land surface modeling, *Journal of Advances in Modeling Earth Systems*, *8*(1), 41–65, 2016.

Rosenblatt, F., The perceptron: a probabilistic model for information storage and organization in the brain., *Psychological review*, *65*(6), 386, 1958.

Rowland, C., D. Morton, L. Carrasco Tornero, G. McShane, A. O'Neil, and C. Wood, Land cover map 2015 (1km percentage aggregate class, gb), 2017.

Salas, F. R., et al., Towards real-time continental scale streamflow simulation in continuous and discrete space, *JAWRA Journal of the American Water Resources Association*, *54*(1), 7–27, 2018.

Scikit-Learn developers, Permutation feature importance, https://scikit-learn.org/stable/modules/permutation_importance.html, accessed: 2020-30-11, 2020.

Vijayakumar, S., The bias–variance tradeoff, *University Edinburgh*, 2007.

Werbos, P. J., Backpropagation through time: what it does and how to do it, *Proceedings of the IEEE*, *78*(10), 1550–1560, 1990.

Wibrow, M., Drawing neural network with tikz, TeX Stack Exchange, uRL:https://tex.stackexchange.com/questions/153957/drawing-neural-network-with-tikz/153974 (version: 2021-01-29), 2014.

Zhao, B., H. Lu, S. Chen, J. Liu, and D. Wu, Convolutional neural networks for time series classification, *Journal of Systems Engineering and Electronics*, *28*(1), 162–169, 2017.

Appendix A

CamelsML documentation

A.1 Installation

Requirements:

- Cuda if using an Nvidia GPU. ROCm should in theory also work if using an AMD Radeon GPU, but needs a specialized version of Pytorch called Pytorch-ROCm. The author has not been successful when trying to run this on an AMD Radeon RX 6800 XT graphics card.
- GNU/Linux (Other operating systems have not been tested)
- Python 3.8
- It is also possible to run machine learning algorithms on CPUs, therefore bypassing the need for CUDA or equivalents, but this is not recommended as it is significantly slower and computationally inefficient to do so.

In the terminal, run:

```
pip install camelsml
```

NB: The author recommends using a dependency resolver better suited than pip. Pipenv (https://pypi.org/project/pipenv/) is what is being used in this thesis. For a Pipfile that is confirmed to work on Ubuntu >= 20.04, as well as an updated installation of Arch Linux at the time this thesis was submitted, see the root of the github repository for this thesis at https://github.com/bernharl/Master-Thesis. CamelsML itself is stored at https://github.com/bernharl/camelsml If you want to install the package using Pipenv, run:

```
pipenv install camelsml --python 3.8
```

```
from camelsml import load_config, train

cfg = load_config(cfg_file="run_config.txt", device="cpu",
↪  num_workers=24)
train(cfg)
```

Listing A.1: Minimal running example of CamelsML

If you run into an error message mentioning Black, try running:

```
pipenv lock --pre

pipenv sync
```

This is caused by the package Black as of the submission date of this thesis not having a stable release that is resolvable with the other packages. Locking with `--pre` allows the use of pre-releases.

A.2 Usage

Here we show a minimum example of how to train an LSTM with CamelsML. The minimal code needed to train an LSTM model is shown in Listing A.1. As seen here, the CamelsML package needs a variable called "cfg", which is a dictionary containing the model configuration, how the train-validation-test split is defined, and so on. A simple example of a configuration file could be this, which trains an ordinary LSTM for 30 epochs using a batch size of 1024 is shown in Listing A.2

For more examples, all the models trained in this thesis, along with scripts for setting up train-test splits and cross validation are contained in the "runs" directory on the Github page for this thesis (https://github.com/bernharl/Master-Thesis/tree/master/runs). If you are reading this some time after the thesis was submitted, there may be updated documentation for CamelsML on the Github page (https://github.com/bernharl/camelsml). As of thesis submission, the released version of CamelsML is version 2.0.2

```
run_dir: <path> # Folder to save runs in
camels_root: <path> # Root folder of dataset
train_start: 01101971 # Date to start training period of
↪ timeseries
train_end: 30092015 # Date to end training period of
↪ timeseries
val_start: 01101971 # Date to start validation period of
↪ timeseries
val_end: 30092015 # Date to end validation period of
↪ timeseries
epochs: 30 # Number of epochs
learning_rate: 1e-3 # Initial learning rate
seq_length: 270 # Sequence length
batch_size: 1024 # Batch size
hidden_size:  256 # Amount of nodes in neural network layers
↪ in the LSTM gates
initial_forget_gate_bias: 5
log_interval: 50 # How often to log
clip_norm: True # Whether to clip gradients
clip_value: 1 # Max of gradient norm
dropout: 0 # Dropout rate
seed: 19970204 # Seed, for reproducability
cache_data: False # Whether to cache all training data in RAM
no_static: True # No static features
evaluate_on_epoch: True # Run evaluation after each epoch
train_basin_file: <path> # Plain text list of basins to use
↪ for training
val_basin_file:  <path> # Plain text list of basins to use
↪ for validation
test_basin_file: <path> # Plain text list of basins to use
↪ for testing
```

Listing A.2: Example configuration file of an LSTM model trained on CAMELS-GB without using static features.