Karlsruher Institut für Technologie

Artificial Neural Networks for Streamflow Prediction

An Introduction to Machine Learning and its Application to Hydrology

Study Project Report
submitted by

Jean-Paul Wißler

Matriculation Number: 1790970

At the Department of Hydrology

Karlsruhe Institute of Technology (KIT) &
Institute for Water and River Management (IWG)

Supervisor: Dr. Ralf Loritz

22nd of February 2022

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst, und weder ganz oder in Teilen als Prüfungsleistung vorgelegt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die benutzten Werken im Wortlaut oder dem Sinn nach entnommen sind, habe ich durch Quellenangaben kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen sowie für Quellen aus dem Internet.

Name: Jean-Paul Wißler

Titel der Arbeit: **Artificial Neural Networks for Streamflow Prediction -** An Introduction to Machine Learning and its Application to Hydrology

Ort, Datum: Karlsruhe den 22.02.2022

Unterschrift: ______________________________

Contents

1. Introduction

The advancements in computational capacity made over the last decades enable scientists nowadays to perform complex and costly computations, for example the numerical simulation of turbulences, and to process vast datasets in a fraction of time than their predecessors. Limitations in computational modelling nowadays rather lie in the model set up. Considering all stationary or dynamic boundary conditions, the physics and state variables, is not only very time intensive but also cannot avoid uncertainties, due to missing data or certain assumptions. In the discipline of hydrology one key challenge is to predict the discharge from hydrological catchments, which is relevant for flood protection, water supply (agricultural, industrial, potable) and for the operation of hydropower plants. State of the art approaches for Rainfall-Runoff modelling include empirical based, conceptual, and physical models. In case of a physical model, it usually does not just take a lot of time to gather the relevant data by surveying the respective catchments and then adjusting a pre-setup physical model, but also to calibrate it, by tuning the models' parameters, to get valid discharge predictions. In the end the derived model usually only has validity for the respective catchment.

With the earlier mentioned now available computational resources is it feasible to use machine learning algorithms, which use up vast amount of memory and perform lots of parallel calculations, for all kind of classification or regression tasks, like rainfall-runoff predictions.

The aim of this study project was to gain extensive insight into the topic of machine learning algorithms, especially on recurrent neural networks, and their application in hydrological Rainfall-Runoff modelling. To achieve this the experiments of the publication "*Rainfall-runoff modelling using Long Short-Term Memory (LSTM) networks*" by Kratzert *et al.*, 2018, were reproduced after getting an overview of the relevant literature.

This report first elaborates the technical specifications and functionalities of machine learning algorithms or more specifically of artificial neural networks (ANN), before presenting the networks architecture used by Kratzert, explaining his experiments, and comparing the results of the reproduction to the original.

2. Theoretical Framework

2.1 Artificial Neural Networks

As a start to the field of artificial intelligence (AI) it is important to give clear distinction between the differences in the nomenclature. Machine learning (ML) labels a subfield of artificial intelligence. ML has various definitions but can be generally defined as a computational algorithm that "learns", without being explicitly programmed, to solve certain tasks by repeatedly processing input data while increasing its performance measure. (El Naqa and Murphy, 2015)

Artificial Neural networks (ANN) are more complex computational learning systems and consist of multiple nodes (neurons) which resemble human brain biology in their functioning, hence their name. They also "learn" to solve task using a series of algorithms and a network of functions. (McClarren, 2021)

Deep neural networks (DNN) are a further extension of ANNs. While a simple ANN, e.g. consists of only 3 layers, namely the input, the hidden and the output layer, a DNN defines an ANN with any number of hidden layers. (McClarren, 2021)

2.1.1 General Structure

2.1.1.1 Input Data

Machine learning algorithms can solve various complex tasks like text interpretation, image classification or regression of timeseries to predict numerical values. This study project is solely interested in the latter which is classified as a supervised learning problem. The datapoints (*samples)* of the timeseries consist of dependent and independent variables. The independent variables also called *features* include state information for an object at a certain point in time. This corresponds to meteorological sensor data in a hydrological catchment. These are later used by the machine learning models or ANNs to predict the dependent variables also known as *labels* or *target values*, in this case represented by a measured discharge value. Supervised learning in this sense means that the model can measure the accuracy of its own prediction by comparing its output to the correct labels, which will be elaborated in the next chapter. (Aggarwal, 2018) ANNs apply parameters to the various independent variables they are fed with. When the features vary in the magnitude of their scales, converging towards a parameter set that yields the best result becomes much slower due to the strong impact of features with high scales and nonlinearity in NNs. For this reason, a certain normalization or standardization is usually conducted on the input data (features and labels). (McClarren, 2021) The z-score is a standard statistical method to do so, by subtracting the mean of the variable from it and dividing it by its standard deviation. (Hedderich and Sachs, 2020)

2.1.1.2 Training

Each model has various parameters[1] (*weights*) which are used for calculating the predictions. These are constantly changed during *training* to fit the data and to optimize the accuracy of the model. The way to achieve this is to calculate the error of the output compared to the target by defining a *loss function* which a certain algorithm (*optimizer*) tries to minimize by adapting the model parameters appropriately. Most commonly used loss function is the mean squared error (MSE). (McClarren, 2021)

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2$$

N	*number of samples*
y	*target value*
$\tilde{y}$	*prediction value*

With multivariate models like the runoff prediction, where there are more than one independent variables, the optimizer needs to be able to, not only find a local minimum of the loss function, which can be achieved by a gradient descent algorithm[2], but the global minimum[3]. A modern algorithm for this non-convex optimization is the *adaptive moment (adam) optimizer*, introduced by Kingma and Ba in 2017. (Kingma and Ba, 2017) It computes adaptive learning rates for each parameter, by considering an (exponentially decaying) average of past squared gradients (of the loss function) and the (also exponentially decaying) average of past gradients. The latter is called momentum since it acts like the momentum of an object going down a slope, which makes sure it does not directly stop

[1] The number of parameters in a model depends on its structure and topology
[2] This algorithm updates the parameters of the model iteratively by following the steepest gradient of the loss function (defined by it first order derivative) till a minimum is reached
[3] A general case for non-linear optimization problems

at the end of the slope (local minimum) but carries on going up again till the momentum, which depends on the gradient of both slopes, is lost or changes due to another downward gradient. (Ruder, 2017) A detailed explanation for the functionality of the different optimizers and their comparison can be assessed in a comprehensive *"Overview of gradient descent optimization algorithms*" by Sebastian Ruder, 2017.

An important *hyperparameter* appearing in combination with the optimizer is the *learning rate*, which is a factor that controls how strongly the parameters are updated in response to the measured error. It also can be seen as the step width of the gradient descent. (Aggarwal, 2018)

Usually, the parameters are adapted after all samples were fed to the network. This cycle is called an *epoch*. But for faster convergence it is common to use *batches* of samples. The parameter update occurs after a whole batch passed through the network. (McClarren, 2021) While batch size[4] is a hyperparameter that affects mainly computational speed, the number of epochs must be well calibrated to avoid over- or underfitting of the data. (Bengio, 2012) An additional way to reduce overfitting is the regularization method called *dropout*, which assigns each node in the network a predefined probability of becoming 0[5] during one sample processing run. This routine prevents strong co-adaption of the neurons. (Srivastava *et al.*, 2014)

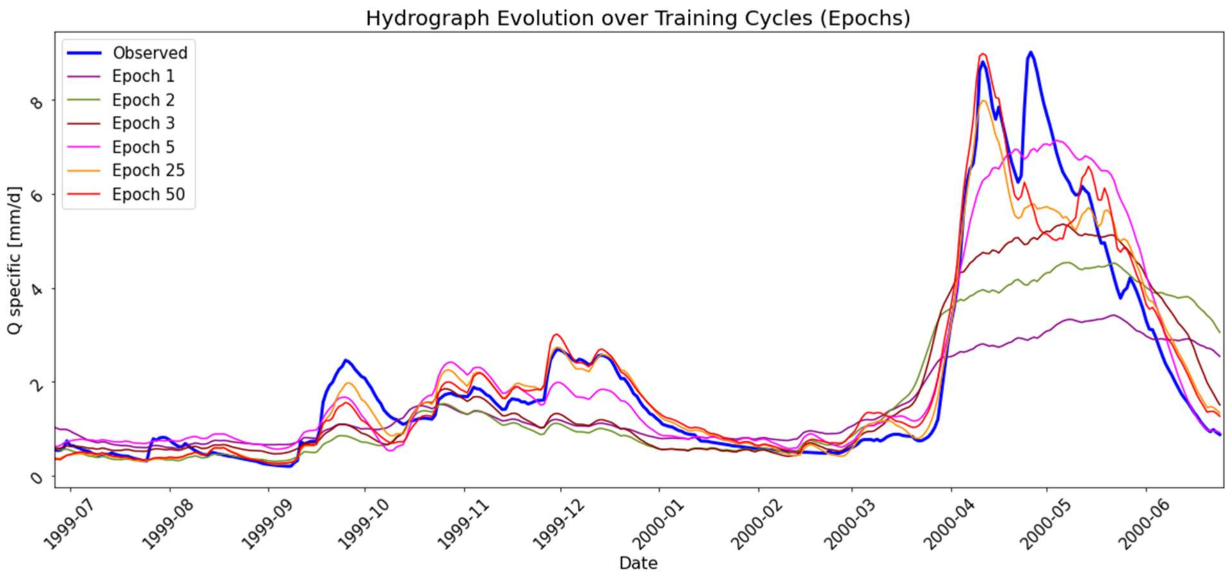
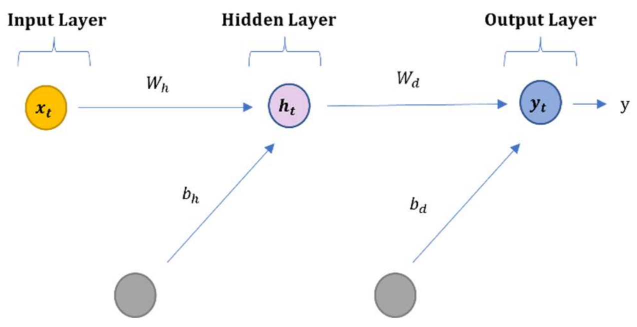


Figure 1 shows the evolution of runoff predictions from a LSTM model for a catchment in the New England HUC.

2.1.1.3 Activation Function

What defines a neural network is the use of neurons in its prediction making. Vast amounts of neurons in several layers can hold the statistical distribution of events dependent on certain boundary conditions. To retrieve the critical information for a single event it is crucial for the network to be able to select or activate only certain neurons, which feed forward the self-processed information. Activation functions serve this purpose by alternating the output of a neuron (pre-activation value) into a, for the network anyway, more interpretable output (post-activation value). There are several different activation functions which are chosen task specific. The *sign* function for example classifies the neurons output into a binary value [0 v 1] while *sigmoid* squeezes it into a smooth bounded function ϵ [0, 1] which value represents the probability of a binary class. (Aggarwal, 2018)

[4] Numbers of samples per batch
[5] Thus, not activating, or "dropping out"



$$h_t = \varphi(W_h x_t + b_h)$$
$$y_t = W_y h_t + b_y$$

Figure 2: Simple ANN layering with feature vector x_t, weight vectors W, biases b and the output vector yt. φ represents any activation function.

2.2 Recurrent Neural Networks

2.2.1 Vanilla Recurrent Neural Network

When training an ANN with sequential data it is important for the network to be aware of the relationship between the different timesteps and not to process each sample individually. By feeding a network, additional to the current input features, with its own output from previous timesteps it can conserve critical information. Such networks, now labelled recurrent neural networks (RNN) were first published in the 80´s and are under constant development, which will be introduced in the next chapters. (Rumelhart, Hinton and Williams, 1986)

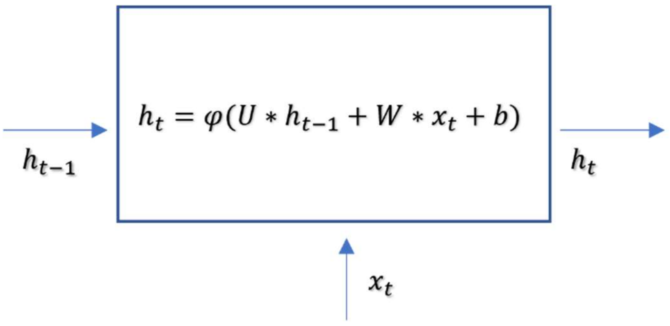


Figure 3 Vanilla RNN Cell

Figure 3 shows a single RNN neuron where the output h for the timestep t is computed, by applying weights U, W to the previous output h_{t-1} and the current input x_t, adding a bias b and wrapping it in an *activation function* ϕ. In a fully connected layer, each node also receives the last hidden state h_{t-1} from each other node in the layer and applies a weight to it. (McClarren, 2021) On handling a *many-to-one*[6] problem as is the case in this project, the network takes an input vector x and the last hidden state for every time step and processes it with the weights in each node to compute a hidden state, which it feeds forward to the next time step, till a prediction is computed in the output layer. This recurs for all samples in a batch before the optimizer backpropagates through the time steps to calculate the gradients of the lost function[7] and then updates the weights.

The recurrent weights for the hidden states become a problem when the sequence is too long, which happens quickly. The short explanation for the so-called *problem of exploding or vanishing gradients* is that the derivative of the error with respect to all the parameters, contains an exponential term for the hidden state weight parameter U^{T-2} where T is the sequence length. If $U < 1$ which is usual when initialising weights before the first training round, U^{T-2} becomes almost zero for a long sequence. This would mean a loss of information from earlier timesteps (vanishing gradients). (McClarren, 2021)

For runoff prediction, this kind of ANN is not suitable considering, that runoff generation in most catchments strongly depends on its meteorological history which influences the soil state, land cover vegetation, snow cover etc., which all have strong impacts on the runoff generation. When for example in spring snow melt kicks in but the network cannot “remember” the due to low temperatures and precipitation, accumulated snow cover of the previous months, it will strongly underestimate the current runoff prediction.

2.2.2 Long Short-Term Memory

Hochreiter and Schmidhuber proposed 1997 a novel method to overcome the lack of a “long-term memory” in RNNs by proposing a new cell architecture which they labelled “Long Short-Term Memory” (LSTM) cell. It adds several mathematical operations to a cell which can catch long term dependencies in sequential input data far better than simple RNNs. (Hochreiter and Schmidhuber, 1997) In contrast to the simple RNN cell, the LSTM cell (FIG) has one additional output, the cell state c_t which serve as the long-time memory and 4 intermediate vectors (i_t, f_t, o_t, $\check{c}_t$). The nonlinear sigmoid function σ squeezes the cell output into a value ϵ [0, 1]. This serves as gating[8], hence forth they will be referred as gates. The forget gate f_t decides if information from the previous cell state c_{t-1} will be deleted, the output gate o_t controls the leakage of information from the previous hidden state h_{t-1} into the current state h_t and the input gate supervises what information gets written into the new memory c_t. To avoid that c_t only increases, the input gate vector is element wise

[6] multiple features to predict one target value

[7] The whole novelty of this algorithm is that it calculates the error by considering previous time steps

[8] See activation function in chapter 2.1.1.3

multiplicated[9] with a memory candidate $\check{c}_t$ which does not use the sigmoid function to regularize its content but uses the $tanh$ function, which is an extension to the former and computes values ϵ [-1, 1]. (Aggarwal, 2018) Only at the end of the cell computation does the memory state affect what will be written as final output h_t, as seen in Figure 4.

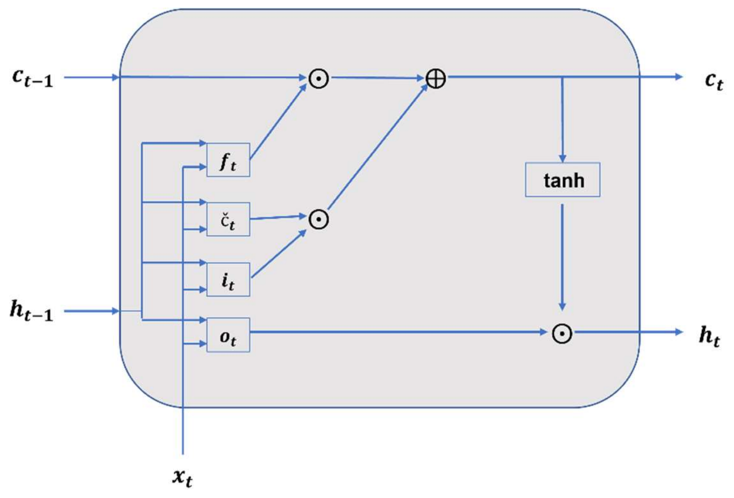


Figure 4 LSTM Cell

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$
$$\check{c}_t = tanh(W h_{t-1} + U x_t + b)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \check{c}_t$$
$$h_t = o_t \circ \tanh(c_t)$$
$$y_t = h_t$$

It is notable that even though the introduction of the memory state revolutionized the performance of RNNs on sequential data, it can easily be observed, that the original cell architecture by Hochreiter and Schmidhuber unnecessarily loses information. By overwriting the previous cell state $c_{t-1}$with the responsible gates without considering its former content in the gating[10] process, the effective memory change only depends on the input x_t and the previous hidden state h_{t-1} but not on itself. This could be solved by introducing additional parameters that scale the previous cell state in the forget and input gate. These "peephole connections" where introduced by Gers *et al.*, in 2002, and will not be further discussed here. Worthwhile mentioning though, it is highly recommended to investigate how this modified architecture can improve the performance of the in the next chapters introduced models. (Gers, Schraudolph and Schmidhuber, 2002)

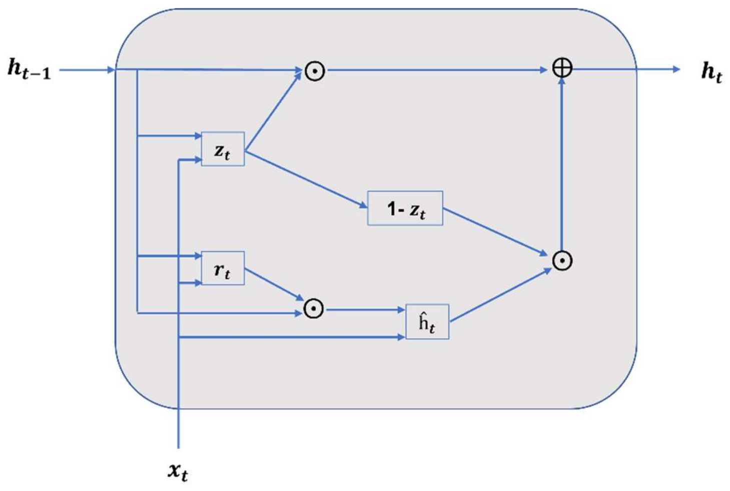
2.2.3 Gated Recurrent Unit

Another sophisticated RNN architecture was described by Cho *et al.* in 2014. Gated recurrent units (GRU) are a further development of LSTM cells and can be seen as a simplified version of the latter, without an explicit cell/memory state. Like the LSTM cell a GRU uses previous hidden states in gate operations to learn long-term dependencies of distinct features. The update gate z_t in the GRU serves the same function as the forget gate in the LSTM, namely conserves important features as they are and the reset gate r_t serves the same function as the update gate. One difference between the two structures besides the lack of a cell state is the controlled way the LSTM exposes its memory states, controlled by the output gate, where the GRU always passes the full content. (Cho *et al.*, 2014)

The two cells also have the input gate, or the corresponding reset gate in different locations. The new memory content in the LSTM cell gets computed without any separated control of the amount

[9] Hadmard produkt
[10] Or filtering for better comprehension

of information passed over from the previous time step. As explained in the previous chapter does the LSTM cell rather control the amount of new memory content being added to the memory cell independently from the forget gate, while on the other hand, the GRU cell is controlling the information flow from the previous activation when computing the new candidate activation without independent control over the amount of the candidate activation being added. Instead, the control is tied up via the update gate. (Chung *et al.*, 2014)



$$r_t = \sigma(W_r h_{t-1} + U_r x_t + b_r)$$

$$z_t = \sigma(W_z h_{t-1} + U_z x_t + b_z)$$

$$\hat{h} = tanh(W(r_t \circ h_{t-1}) + U x_t + b)^{11}$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \hat{h})$$

Figure 5 GRU Cell

From these differences and similarities alone, it is difficult to deduce which architecture performs better in the end. For this reason, both are applied to the models in the following experimental designs.

2.3 Machine Learning in Hydrology

ANNs are merely a modern form of *data driven models* and their application to hydrology, especially streamflow prediction has been around for over two decades. Early approaches to model hydrological runoff with ANNs included several different architectures (e.g. (Hsu, Gupta and Sorooshian, 1995); (MINNS and HALL, 1996)) till arriving at RNNs to be the seemingly most promising one (e.g. (Pan and Wang, 2005)). The hydrological scientific community seems to be divided though, in regards how to view such models which superficially lack any physical representations and heavily depend on their training sets. While the initial assessment of a critical standpoint towards a "black box" in data driven models or ANNs seems legit, the disregard of other modelling approaches than the one promoted by oneself, is unfortunately common practice for hydrological modellers. (Todini, 2007)

The in this study project used paper and its reproduced findings, mark a turning point for hydrological runoff prediction. A multinational group of scientists surrounding the core authors, Frederick Kratzert, Martin Gauch, Grey S. Nearing, Daniel Klotz and several others from both national and public research institutions, since published a series of papers, which findings make LSTM networks *" […] currently the most accurate and extrapolatable streamflow models available from the hydrological science community.".* (Nearing *et al.*, 2021)

[11] The original equation from Cho et al. differs slightly from the in Tensorflow implemented version represented here

They address and refute a common concern among hydrologists, that data-driven deep learning models are unreliable in extrapolation or for predicting extreme events (floods) (Frame *et al.*, 2021), tackle the regional modelling problem by outperforming conceptual and physically-driven benchmark models for individual basins with regional LSTM models (Kratzert *et al.*, 2018), form strategies for near-real time streamflow prediction with LSTMs (Nearing *et al.*, 2021) and try to show that the LSTM models actually deduce the catchments physics from the input data (Lees *et al.*, 2021).

Even though the latter statement needs to be treated with care since the related paper is still under review, there already is some strong evidence that the LSTMs can "learn" the relation between low temperatures and snow accumulation. (Kratzert *et al.*, 2018) However, explaining the (not so) ominous black box to decision makers remains a sensible topic and hydrological modellers should treat with care when attributing ANNs with more inherent understanding of catchment physics than they can prove.

3. Rainfall-Runoff Modelling

Kratzert et. al, show in their study *"Rainfall-runoff modelling using Long Short-Term Memory (LSTM) networks"* a promising application of machine learning algorithms in hydrology. Using a LSTM architecture, they modelled 241 catchments in the United States of America by using up to 35 years of daily meteorological data samples, which are freely available in the CAMELS (Addor *et al.*, 2017) data set. They compared the results to the well-known Sacramento Soil moisture Accounting Model (SAC-SMA) coupled with the Snow-17 (Newman *et al.*, 2015) snow routine and arrived at comparable performance metrics as the benchmark model without performing extensive hyperparameter calibrations. (Kratzert *et al.*, 2018)

This project aimed at reproducing the results presented in the study by rebuilding the LSTM networks presented in Kratzerts paper and training them with the same samples from the CAMELS dataset. After explaining the different experimental set-ups, the results of this project will be compared to the results of Kratzert and possible differences discussed. Additionally, to the networks from the study a set of GRU based networks were trained and evaluated in the same manner and with the same hyperparameters to see if they can outperform the LSTM networks.

The programming environment used for the project was COLAB, a free cloud service by google, that provides GPU and CPU capacities for machine learning applications. (Bisong, 2019) Programming language was the open-source software Python 3.6 (Van Rossum and Drake Jr, 1995). Libraries used were, Pandas (McKinney, 2010), Numpy (Harris *et al.*, 2020), Glob and Pydrive for data handling and Matplotlib (Hunter, 2007) for plotting figures. Deep-learning frameworks were Keras (Chollet and & others, 2015) and Tensorflow (Abadi *et al.*, 2016).

The projects source code can be accessed via a Git Hub repository:

https://github.com/JapeTheEternalChild/LSTM-Streamflowprediction-with-Keras

3.1 Dataset

While the CAMELS dataset includes meteorological data from multiple sources (daymet, maurer, nldas) only the daymet set was used since it got the highest spatial resolution (1km grid). It includes daily mean values for precipitation (mm/d), shortwave radiation (W/m²), max and min temperatures

(C°), vapor pressure (Pa), daylight (s) and snow water equivalent (mm), for 671 catchments which are grouped into hydrological units (HUCs) after the U.S. Geological Surveys (USGS) HUC map. (Seaber, Kapinos and Knapp, 1987) The latter 2 variables are excluded in the experiments, the others pose the features for the ANN. Seconds of daylight per day is constant and periodic, thus has little to no relevant information for the network to learn from and snow covering is 0 during the most time of the year and in some catchments during the whole year, which also means a lack of information.[12] Furthermore, daily streamflow data (feet^3/s) from the USGS is included in the dataset and serves as target values. A parametric standardisation (z-score) is conducted on the variables after splitting them into a test and a training data set. Only the statistical values (mean & standard deviation) from the training dataset are used for standardisation.[13] The split follows Kratzert using the period from 1981-10-01 to 1995-09-30 for training and the period 1995-10-01 to 2010-10-01 for testing the models. He decides against the widespread calibration strategy of splitting the data into training, calibration, and validation, where the first two periods are used to calibrate the parameters of the network and the last to measure its performance, so he can compare his results to Newman *et al*. (Newman *et al.*, 2015).

For the network to be able to process the data it needs the data to be passed in tensor form. Tensors are multidimensional arrays of numerical values. (Rabanser, Shchur and Günnemann, 2017) In this case the shape of the input tensors is *(samples, timesteps, features)* and the target tensors *(samples, labels).* For every prediction the networks make, a sample with 5 features for the current time step and the last 365 days (timesteps) preceding it are passed over.

[12] Kratzert showed how the cell states act as a storage which accumulates with low temperatures during winter and release its storage when temperatures rise again in spring. Thus, deduces that the network "learned" a simple physical relation between precipitation and temperature and its effect on runoff generation.

[13] This method assumes static statistics and should be revised if trends in the data exists (e.g., raising mean temperatures → increased humidity → more annual precipitation) and subsequently mean and standard deviation between train and test set strongly differ from each other.

3.2 LSTM Network

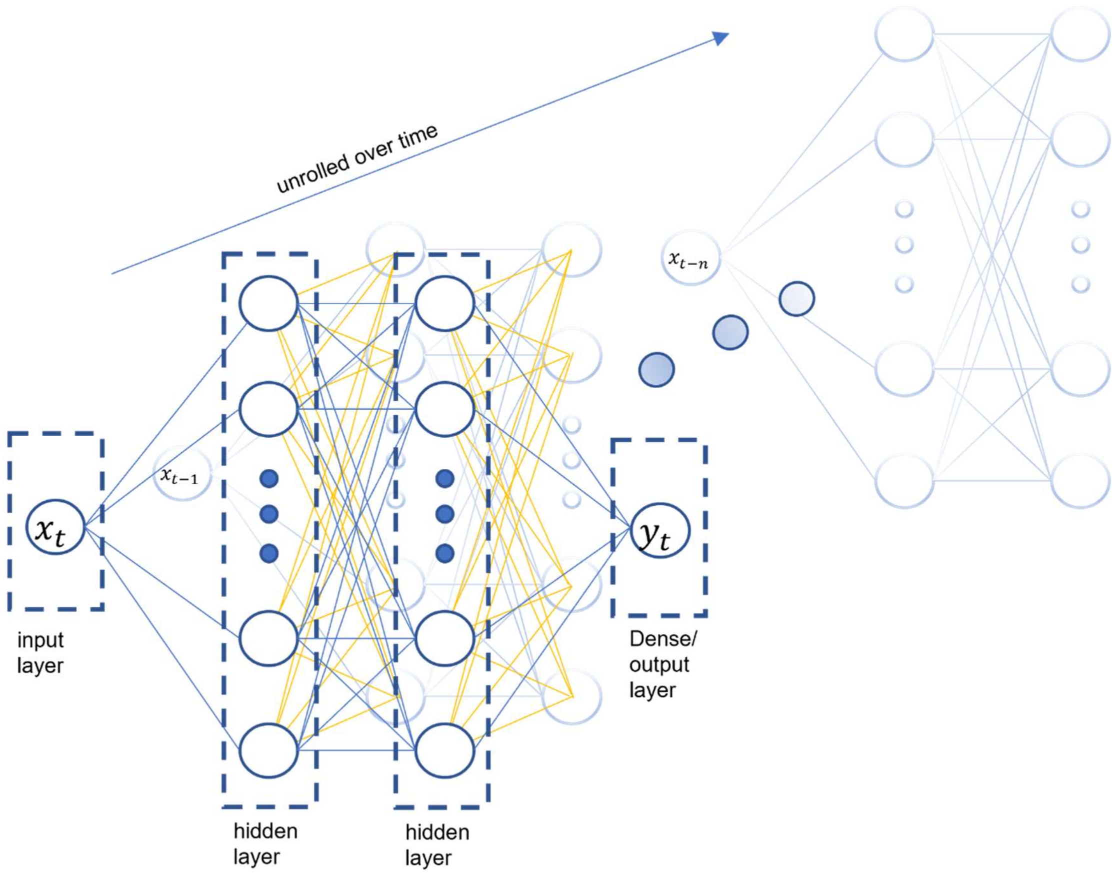


Figure 6 LSTM network unrolled over time: The figure shows the networks topology and its information flow when predicting a single value. Starting with the sample xt-n the features are passed through the hidden layers. (pass visualized by blue lines) In the next computation step xt-(n-1) all hidden states from the respective layer in the previous computation are passed over to each node. (pass visualized by yellow lines). This recures till the end of the sequence when xt is processed and an output value is computed in the dense layer.

Figure 6 shows how one prediction is derived by the network, by unrolling it over time. The network first processes the n last timesteps contained in the input tensor before arriving at a value for the current one which then is used to calculate the loss against a corresponding label in the target tensor. Usually, all samples are passed through the network before the optimizer adapts the parameters (weights) dependent on the loss function gradients. Though for faster convergence of the weights, batches are defined which contain 512 consecutive samples. The weight update now occurs after one batch went through the network. The number of epochs defines how often all samples, collected in batches, are passed through. The batches themselves are not in consecutive order. It is only of importance to preserve the order the samples contained within them. All hyperparameters are listed in Table 1. Table 2 gives an overview of the different network layers and their respective parameters (weights).

Table 1 Hyperparameters

Hyperparameter	Value/Metric
Epochs	50
Batch size	512
Sequence length	365d
Learning rate	1e-3 (Adam)
Loss	MSE
Dropout	0.01
Hidden layer	2
Hidden layer size	20

Table 2 Model Summary

Layer (type)	Output shape	Parameter #
Hidden 1 (LSTM)	(None, 365, 20)	2080
Hidden 2 (LSTM	(None, 20)	3280
Output (Dense)	(None, 1)	21
Total # of trainable parameters:		5381

3.3 Experimental Design

3.3.1 Experiment 1

The first experiment consists of training and testing one model for each of 241 catchments from 4 different HUCs with broadly varying climatic conditions and catchment characteristics. In a preliminary experiment by Kratzert the optimal number of epochs was derived to be 50. To conduct this, the 15 years of the training set was further split into a 14-year training and a 1-year independent validation period. After training all 241 models for 200 epochs, Kratzert validated each model after each epoch with the validation period to find the number of epochs that result on average in the highest Nash-Sutcliffe Efficiency (MNSE) (Nash and Sutcliffe, 1970). Primary evaluation metric remains the NSE.

$$NSE = 1 - \frac{\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2}{\sum_{i=1}^{N}(y_i - y_{mean})^2}$$

3.3.2 Experiment 2

In the second experiment the meteorological data from all catchments within a HUC is bundled together and used to train one model for each HUC. Afterwards the models are applied to the individual catchments within their respective HUCs for runoff predictions.[14] Since more samples result in more batches and thus more weight updates per epoch, the overall number of epochs was reduced. The optimal amount of 20 epochs was determined by Kratzert in the same preliminary experiment as in Experiment 1.

3.3.3 Experiment 3

Finally, the last experiment takes the respective models from the second experiment to serve as *pre-trained* models for the individual catchments from the respective HUCs. Meaning the general patterns[15] derived from the data fed to the models in Experiment 2 are already represented in the weights of the same. Using a pretrained model for further training on smaller datasets is known as *fine-tuning*. (Razavian *et al.*, 2014) (Yosinski *et al.*, 2014) Kratzert trains each catchment model for an individual number of epochs ranging from 0 to 20, to obtain the best NSE. In this project each model was trained for a fix number of 10 epochs.

[14] i.e., using the test set of the catchment on the model trained for the respective HUC
[15] Relationship between meteorological features and the streamflow target values

3.4 Results & Discussion

Since the result of Kratzerts study are not publicly available but the ones from the benchmark study by Newman (Newman *et al.*, 2015) are, only the latter are quantitatively compared to the results of this study project. Qualitative remarks to Kratzert study relate to the metrics published in his paper. (Kratzert *et al.*, 2018)

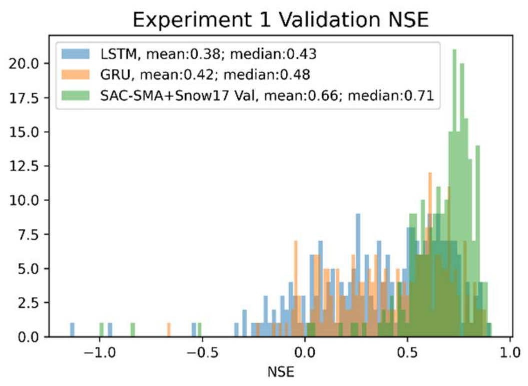


Figure 7: Histogram showing the models performances of Experiment 1 against SAC-SMA+Snow17

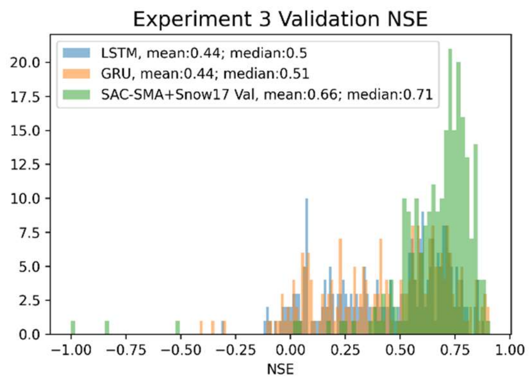


Figure 8: Histogram showing the models performances of Experiment 3 against SAC-SMA+Snow17

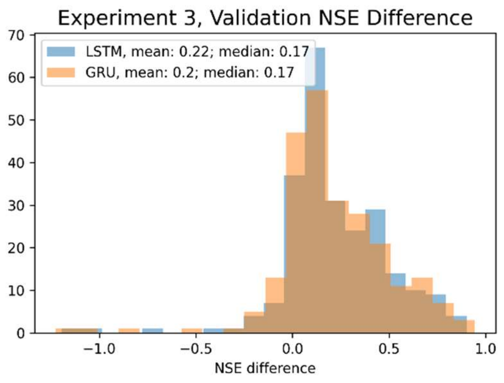


Figure 9: Histogram showing the performance differences for the catchment models from Experiment 3 against the SAC-SMA+Snow17

The overall performance metrics of the Experiments fall far off the ones published by Kratzert, which are close to the benchmark model evaluation metrics.

The GRU based models show slightly better results in Experiment 1 (Figure 7) but perform worse in the second one (Appendix VII:), which ultimately results in almost equal metrics in Experiment 3 (Figure 8).

It appears, that the LSTM models are not consistent in their performances. While some models achieved a better NSE value compared to the benchmark model, the mean differences in Experiment 3, which yielded the best results due to the pre-trained models from Experiment 2, is 0.22. (Figure 9) Kratzert achieved a median NSE of 0.72 and thus performed comparably well to the benchmark model with 0.71, *"while the mean for the models of Experiment 3 is about 15% higher (0.68 compared to 0.58)".* Figure 8 shows the metrics for the benchmark model. A median of 0.71 and mean of 0.66, not 0.58.[16] The project experiments did not result in any of these.

Several reason for this can be identified when looking at the LSTM models hyperparameters and their impacts on the model performance. First, some of the difference in Experiment 3 can be traced back to the individual epoch calibration Kratzert conducted in his study (ranging from 0 to 20) while in this project a fix epoch number of 10 was used. But then there are already huge dissimilarities in Experiment 1 where Kratzert reports a mean NSE of 0.63 over all catchments, compared to 0.38 in this project. Bengio (Bengio, 2012) gives in his "*Practical recommendations for*

[16] Since his modelling approach needs 365 days of meteorological data for predicting the first timestep, his test set lacks prediction for the first year. He recomputed all metrics for the benchmark model for the same simulation period as his test period.

gradient-based training of deep architectures" an overview of the different hyperparameters involved in deep ANNs and how they affect their performance. He emphasizes the strong impact of the initial parameter set within small ANNs[17]. Kratzert mentions this in his paper but discards the practice of running the same catchment model multiple times to find the random seed with the best model realisation[18]. Simply running the experiments in this project, a second time and differentiating their performance metrics from the previous ones resulted in validation NSE variances for the individual catchments of up to 0.4 (one outlier of 0.8) in Experiment 1 and up to 0.3 in Experiment 3. (Appendix VI:) Another possible reason for the divergence between the projects and Kratzert results could be a different learning rate in the optimizer. Kratzerts paper does not mention the learning rate at all, which after Bengio is possibly the most important hyperparameter of all. When using the Adam optimizer with Keras, a default learning rate of 10e-3 is set. Since there is no information in the paper about the learning rate, this project experiments used the default. But when training random models in Experiment 1 with different learning rates, the NSE could be improved by up to 0.18.

However, even though Kratzerts results could not be exactly reproduced, there are models with a good fit (Experiment 3, NSE ~20% >0.7) (Figure 10) to the test data and even the models with a bad fit catch the overall dynamic of the basins runoff generation. Additionally, like the original study do the models perform better in catchments with snow influence and higher mean annual precipitation (i.e., New England & Pacific Northwest). Vice versa deteriorates the performance of models in the more arid catchments. (Appendix I:, ff.) Kratzert already noted that the LSTM networks seem to have difficulties to deduce information from constant discharge values of zero in a high percentage of the training samples. Nevertheless, does the benchmark model have similar dips in its performance when predicting the discharge of arid catchments. There is no direct link between precipitation-discharge correlation and model performance found in this project [APX]. Further analysis of the input data did not reveal any insight on why some models performed much worse than its peers[19]. However, it was again possible to improve them significantly (NSE +0.45) by changing the various other hyperparameter. These changes range from increasing the number of epochs manyfold to reducing the sequence length from a year to a month.

[17] The number of nodes in the here used LSTM (40) is rather small compared to bigger ANNs with thousands of neurons.

[18] He argues that it would make it more difficult to compare the results thus to the benchmark model. Which is confusing since Newman (Newman *et al.*, 2015) himself calibrated 10 models per basin, with different random seeds to find the model with the best performance metric.

[19] Considering similar catchment characteristics

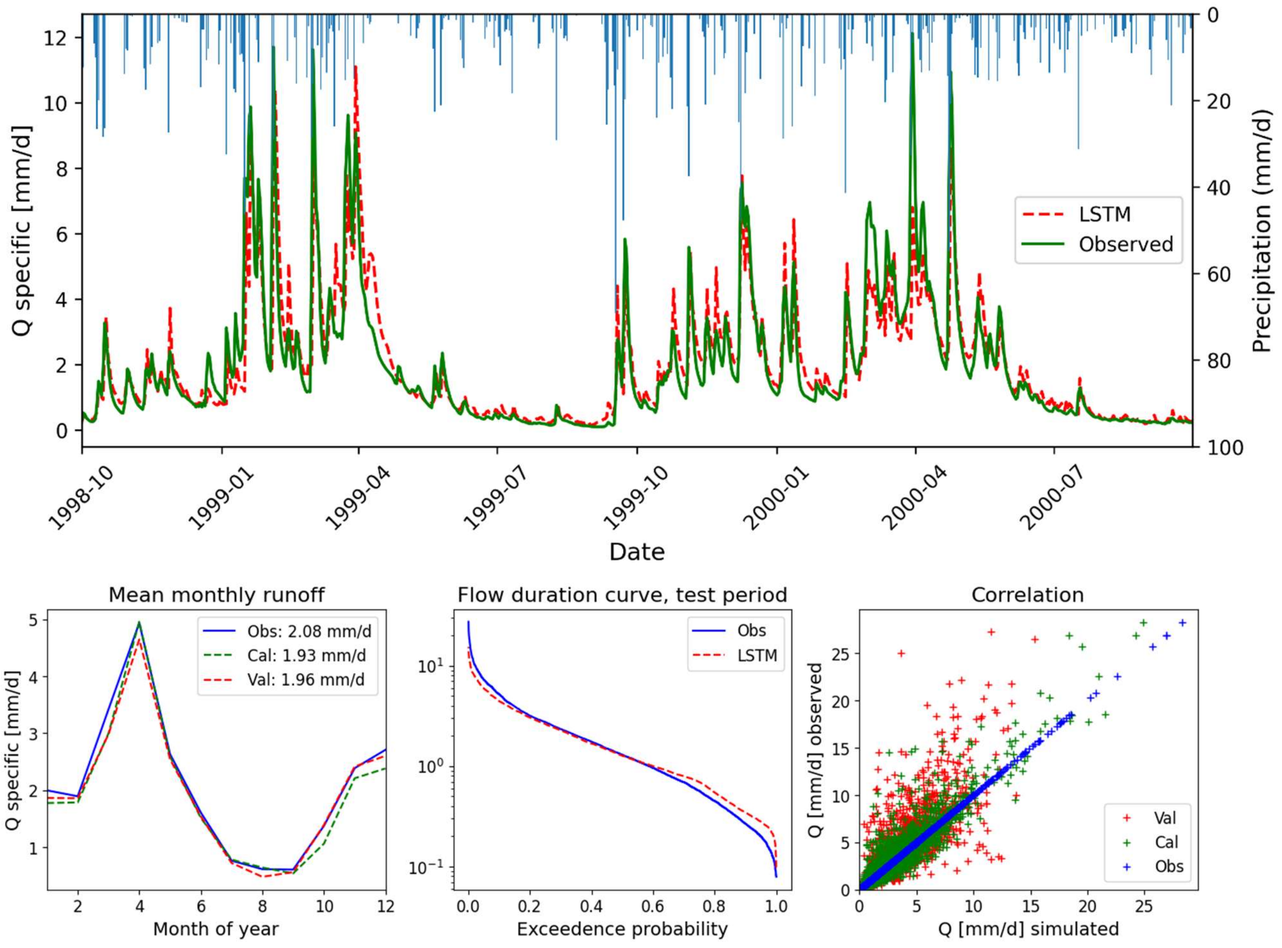


Figure 10: The top panel shows the hydrograph for Catchment 01022500 located in the New England HUC 01 with a simulated sequence in the test period (NSE=0.70) from Experiment 3 against the observed runoff values. The panel on the bottom left shows how model tends to slightly underestimate the mean monthly runoff generation but can catch its annual dynamics. The flow duration curve shows the models incline to overestimate base flow while underestimating high runoff events. The correlation panel emphasizes this underestimation while showing a good fit for the training period (0.86), which indicates overfitting. This relates to the fix number of 10 epochs used in this project unlike in the original experiment conducted by Kratzert with individual epochs for the finetuning. Running the fine-tuning process for only 6 epochs results in an NSE of 0.82.

4. Conclusion

First, even though my project results fall short of the ones published by Kratzert, they are still remarkable. 241 catchments with partly highly varying climatic conditions and catchment characteristics were modelled basically with the same (hyper-) parameter set and the models performed reasonably well in the mean. I showed that the performance of single models also can significantly be improved by tailored hyperparameter adaption. There are automated means and methods to do so with minimal efforts, e.g., using a grid search for a pre-build model or by using *Auto ML* by Google which also helps to find the best network topography (layer, nodes etc.) in advance to the hyperparameter tuning but manual hyperparameter tuning also quickly improve performance. This implicates a high margin for improvements of the individual catchment models.

Also, the introduction of additional features[20] into the network may improve runoff predictions even further. While the consideration of new features in conventional models needs a conceptualization of the same or the implementation of their physical relation into the existing model, an ANN simply needs another filled column in their input data set.

The merits of ANNs are overwhelming but they heavily rely on large datasets to learn from, which are not always available. Preparing the data normally takes far more time than the modelling process itself and gets more complicated with different procurable data products. Furthermore, the quality of the products must be checked, since ANNs are sensitive to sample errors in the training process. The problem of the internal black box remains to some degree. Especially when presenting modelling results to decision makers, the public or even fellow hydrological modellers it is easier to communicate the physics and concepts of conventional hydrological models, that enabled them to make certain prediction, than to explain how the ANN model "learned" how a catchment reacts to meteorological conditions. But efforts are made to make this more graspable, and it is possible to read out the internal states of the network during training and afterwards which can then be linked to certain features and their correlation.

Machine learning algorithms are nothing new, they have been around for decades. But their introduction into the various scientific disciplines especially hydrology, is only happening gradually. Kratzert and his team showed beautifully how the concept of ANNs or more specifically LSTM networks can be easily used with open-source software and a large (also open) dataset to train models which can predict catchment runoffs similar or even better than a more complex conceptual benchmark model.

As a downturn it needs to be mentioned that the lack of any published code material or the raw results by the modellers make it difficult to exactly identify errors or deviations in the reproduction. Subsequent publications mentioned in chapter 2.3 do include this in form of linked repositories on Git Hub with complete source codes. This means a huge gain not only for the scientific community but for society in general, especially when considering the open-source character of the programming language, libraries and datasets used in the studies. Making the full extend of the code used public enables ultimately more people to participate in its further development and make the results reproducible. In this spirit the code used is also available on a Git Hub repository (p. 12) including this report to make it more accessible and understandable.

I hope with this report I have given any reader a decent and narrow introduction into the vast field of machine learning, with focus on LSTMs and their application to hydrology, in the same way I gained insight to it during this study project. There is much more to explore with this powerful and flexible tool and I am already looking forward to so.

"*We can only see a short distance ahead, but we can see plenty there that needs to be done.*"

(Turing, 1950)

[20] Soil moisture, landcover (one-hot encoding) or the like

References

Abadi, M. *et al.* (2016) ‘TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems’, *arXiv:1603.04467 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1603.04467 (Accessed: 21 February 2022).

Addor, N. *et al.* (2017) ‘The CAMELS data set: catchment attributes and meteorology for large-sample studies’, *Hydrology and Earth System Sciences*, 21(10), pp. 5293–5313. doi:10.5194/hess-21-5293-2017.

Aggarwal, C.C. (2018) *Neural Networks and Deep Learning: A Textbook*. Cham: Springer International Publishing. doi:10.1007/978-3-319-94463-0.

Bengio, Y. (2012) ‘Practical recommendations for gradient-based training of deep architectures’, *arXiv:1206.5533 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1206.5533 (Accessed: 13 January 2022).

Bisong, E. (2019) ‘Google Colaboratory’, in Bisong, E. (ed.) *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, pp. 59–64. doi:10.1007/978-1-4842-4470-8_7.

Cho, K. *et al.* (2014) ‘Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation’, *arXiv:1406.1078 [cs, stat]* [Preprint]. Available at: http://arxiv.org/abs/1406.1078 (Accessed: 26 December 2021).

Chollet, F. and & others (2015) *Keras: Deep Learning for humans*. Keras. Available at: https://github.com/keras-team/keras (Accessed: 21 February 2022).

Chung, J. *et al.* (2014) ‘Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling’, *arXiv:1412.3555 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1412.3555 (Accessed: 26 December 2021).

El Naqa, I. and Murphy, M.J. (2015) ‘What Is Machine Learning?’, in El Naqa, I., Li, R., and Murphy, M.J. (eds) *Machine Learning in Radiation Oncology: Theory and Applications*. Cham: Springer International Publishing, pp. 3–11. doi:10.1007/978-3-319-18305-3_1.

Frame, J. *et al.* (2021) ‘Deep learning rainfall-runoff predictions of extreme events’, *Hydrology and Earth System Sciences Discussions*, pp. 1–20. doi:10.5194/hess-2021-423.

Gers, F.A., Schraudolph, N.N. and Schmidhuber, J. (2002) ‘Learning Precise Timing with LSTM Recurrent Networks’, *Journal of Machine Learning Research*, 3(Aug), pp. 115–143.

Harris, C.R. *et al.* (2020) ‘Array programming with NumPy’, *Nature*, 585(7825), pp. 357–362. doi:10.1038/s41586-020-2649-2.

Hedderich, J. and Sachs, L. (2020) *Angewandte Statistik: Methodensammlung mit R / Jürgen Hedderich, Lothar Sachs*.

Hochreiter, S. and Schmidhuber, J. (1997) ‘Long Short-term Memory’, *Neural computation*, 9, pp. 1735–80. doi:10.1162/neco.1997.9.8.1735.

Hsu, K., Gupta, H.V. and Sorooshian, S. (1995) 'Artificial Neural Network Modeling of the Rainfall-Runoff Process', *Water Resources Research*, 31(10), pp. 2517–2530. doi:10.1029/95WR01955.

Hunter, J.D. (2007) 'Matplotlib: A 2D Graphics Environment', *Computing in Science Engineering*, 9(3), pp. 90–95. doi:10.1109/MCSE.2007.55.

Kingma, D.P. and Ba, J. (2017) 'Adam: A Method for Stochastic Optimization', *arXiv:1412.6980 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1412.6980 (Accessed: 3 January 2022).

Kratzert, F. *et al.* (2018) 'Rainfall–runoff modelling using Long Short-Term Memory (LSTM) networks', *Hydrology and Earth System Sciences*, 22(11), pp. 6005–6022. doi:10.5194/hess-22-6005-2018.

Lees, T. *et al.* (2021) 'Hydrological Concept Formation inside Long Short-Term Memory (LSTM) networks', *Hydrology and Earth System Sciences Discussions*, pp. 1–37. doi:10.5194/hess-2021-566.

McClarren, R.G. (2021) *Machine Learning for Engineers: Using data to solve problems for physical systems / by Ryan G. McClarren*.

McKinney, W. (2010) 'Data Structures for Statistical Computing in Python', *Proceedings of the 9th Python in Science Conference*, pp. 56–61. doi:10.25080/Majora-92bf1922-00a.

MINNS, A.W. and HALL, M.J. (1996) 'Artificial neural networks as rainfall-runoff models', *Hydrological Sciences Journal*, 41(3), pp. 399–417. doi:10.1080/02626669609491511.

Nash, J.E. and Sutcliffe, J.V. (1970) 'River flow forecasting through conceptual models part I — A discussion of principles', *Journal of Hydrology*, 10(3), pp. 282–290. doi:10.1016/0022-1694(70)90255-6.

Nearing, G.S. *et al.* (2021) 'Technical Note: Data assimilation and autoregression for using near-real-time streamflow observations in long short-term memory networks', *Hydrology and Earth System Sciences Discussions*, pp. 1–25. doi:10.5194/hess-2021-515.

Newman, A.J. *et al.* (2015) '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), pp. 209–223. doi:10.5194/hess-19-209-2015.

Pan, T. and Wang, R. (2005) 'Using recurrent neural networks to reconstruct rainfall-runoff processes', *Hydrological Processes*, 19(18), pp. 3603–3619. doi:10.1002/hyp.5838.

Rabanser, S., Shchur, O. and Günnemann, S. (2017) 'Introduction to Tensor Decompositions and their Applications in Machine Learning', *arXiv:1711.10781 [cs, stat]* [Preprint]. Available at: http://arxiv.org/abs/1711.10781 (Accessed: 21 January 2022).

Razavian, A.S. *et al.* (2014) 'CNN Features off-the-shelf: an Astounding Baseline for Recognition', *arXiv:1403.6382 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1403.6382 (Accessed: 21 February 2022).

Ruder, S. (2017) 'An overview of gradient descent optimization algorithms', *arXiv:1609.04747 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1609.04747 (Accessed: 3 January 2022).

Rumelhart, D., Hinton, G.E. and Williams, R.J. (1986) 'Learning internal representations by error propagation', in. doi:10.1016/B978-1-4832-1446-7.50035-2.

Seaber, P.R., Kapinos, F.P. and Knapp, G.L. (1987) *Hydrologic unit maps*, *Hydrologic unit maps*. USGS Numbered Series 2294. U.S. G.P.O.,. doi:10.3133/wsp2294.

Srivastava, N. *et al.* (2014) ‘Dropout: A Simple Way to Prevent Neural Networks from Overfitting’, *Journal of Machine Learning Research*, 15(56), pp. 1929–1958.

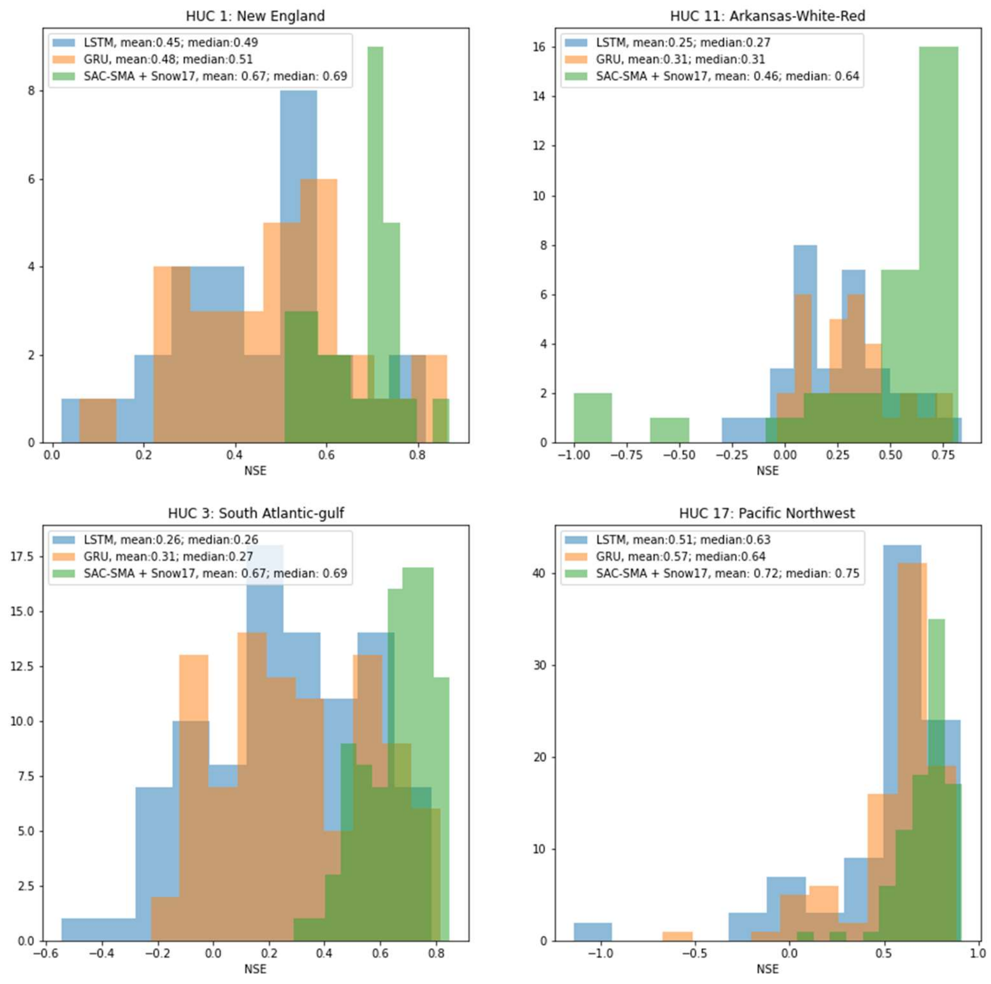
Todini, E. (2007) ‘Hydrological catchment modelling: past, present and future’, *Hydrology and Earth System Sciences*, 11(1), pp. 468–482. doi:10.5194/hess-11-468-2007.

Turing, A.M. (1950) ‘Computing Machinery and Intelligence’, *Mind*, 59(October), pp. 433–60. doi:10.1093/mind/LIX.236.433.

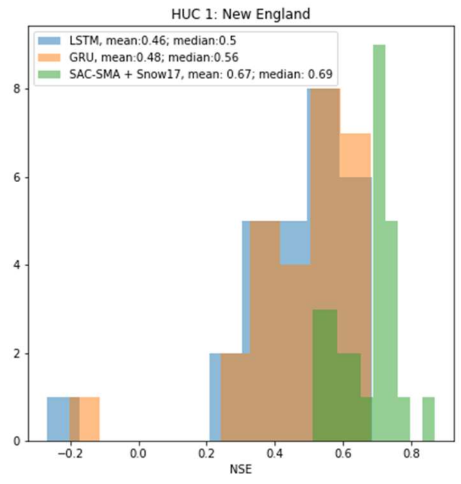
Van Rossum, G. and Drake Jr, F.L. (1995) *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam.

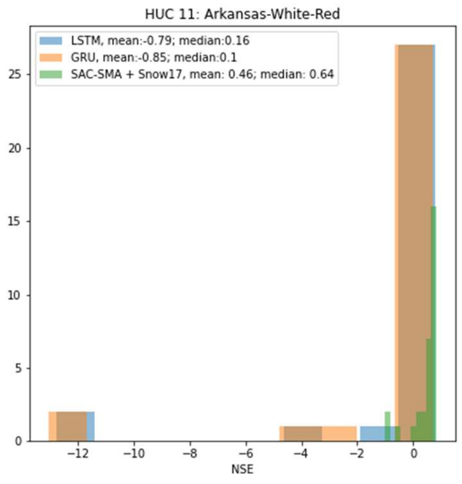
Yosinski, J. *et al.* (2014) ‘How transferable are features in deep neural networks?’, *arXiv:1411.1792 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1411.1792 (Accessed: 21 February 2022).

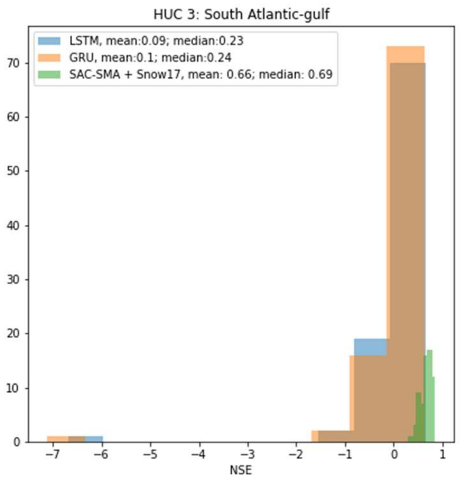
Appendix I: Model performance catchment wise: Experiment 1

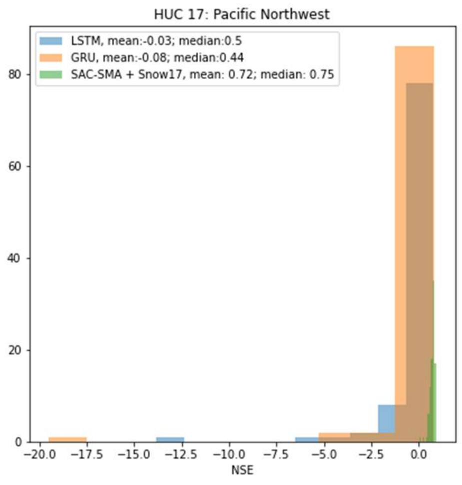


Appendix II: Model performance catchment wise: Experiment 2

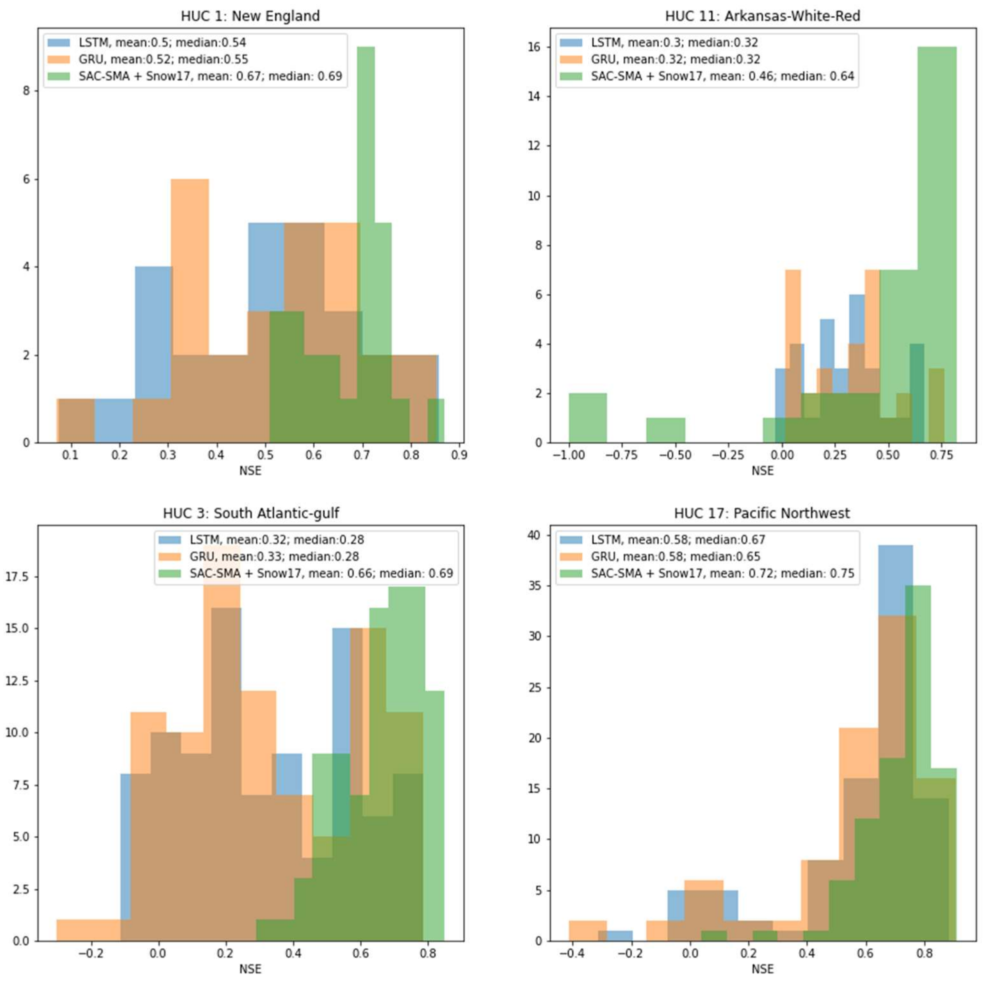




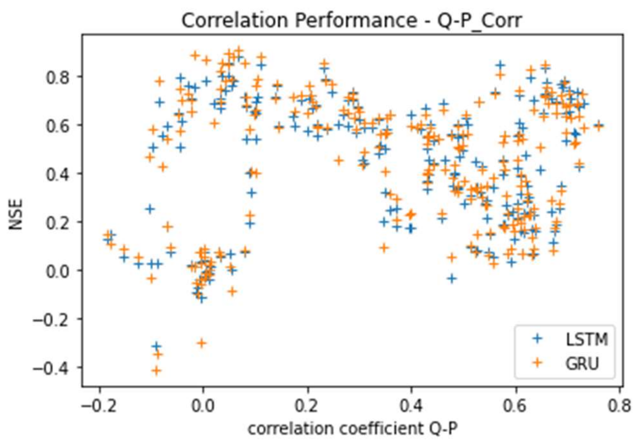




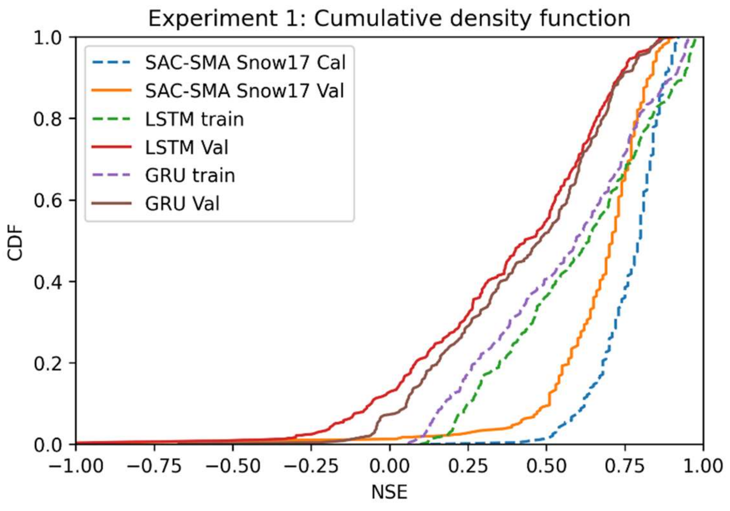
Appendix III: Model performance catchment wise: Experiment 3

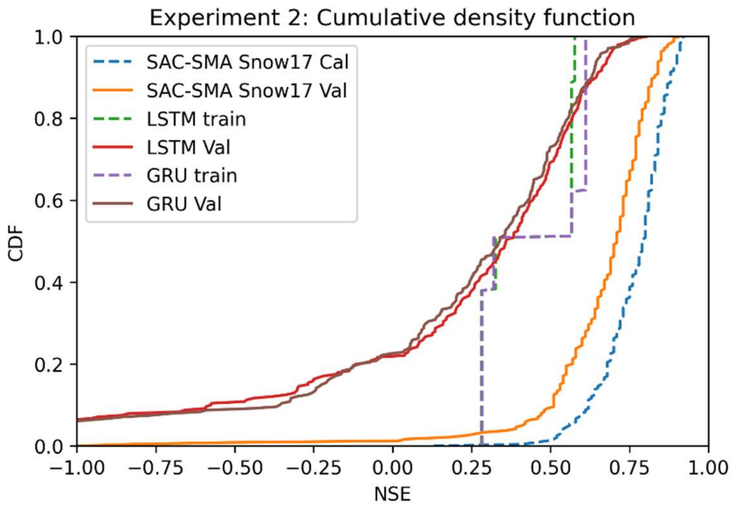


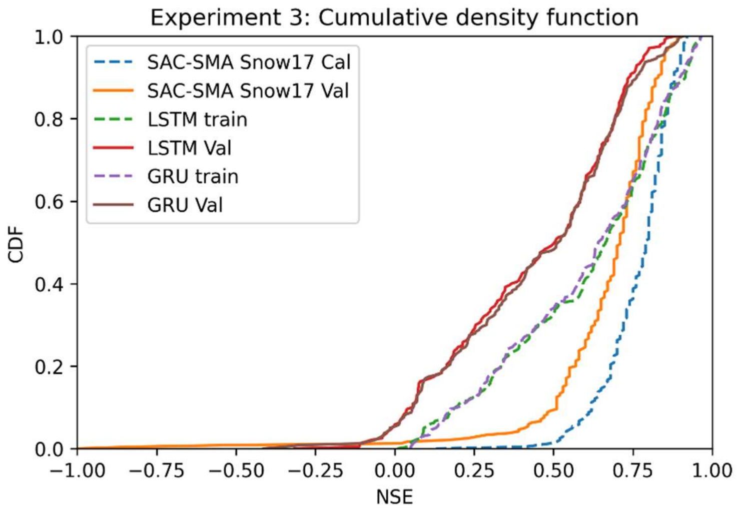
Appendix IV: Performance correlation with Q-P correlation: Experiment 3



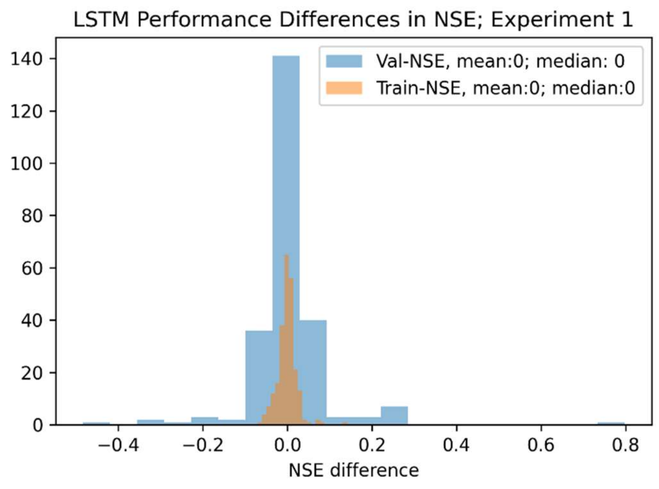
Appendix V: Cumulative Distribution Curve – Performance

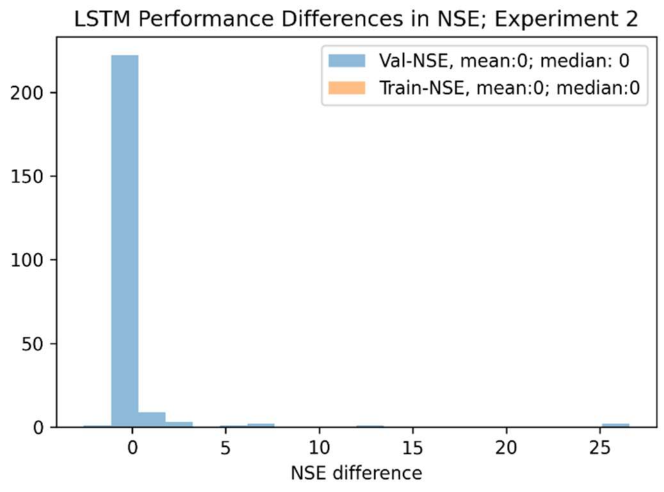


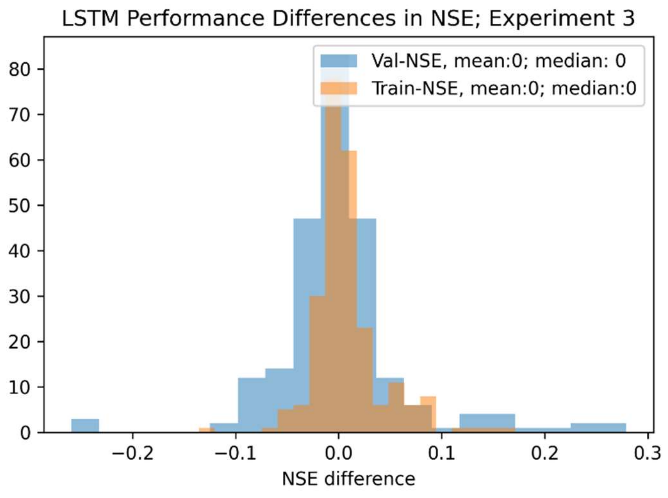




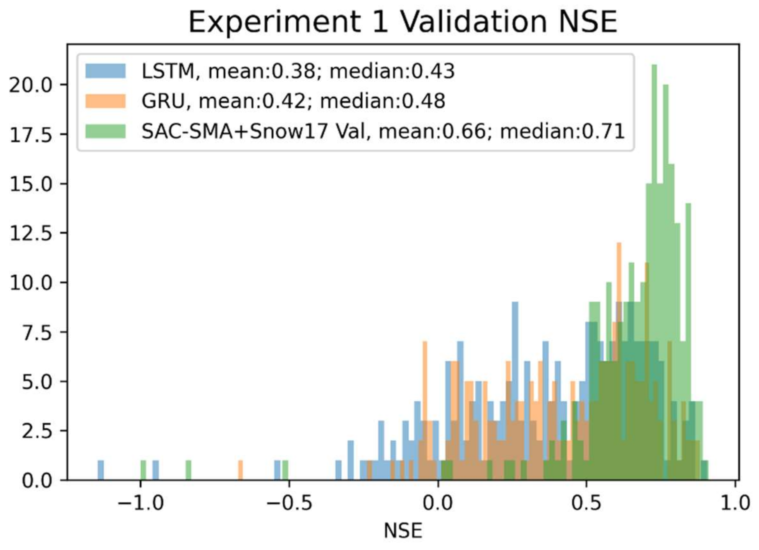
Appendix VI: Performance for different random seeds (weights)

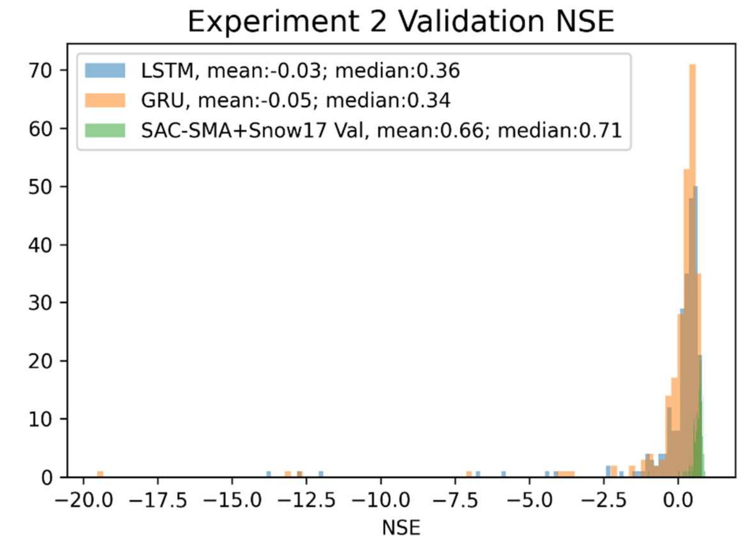


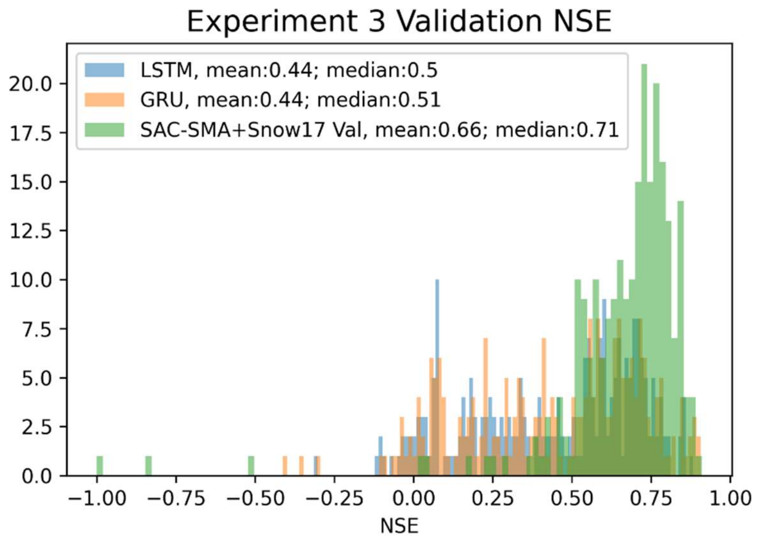




Appendix VII: Validation NSE


Experiment 1 Validation NSE
LSTM, mean:0.38; median:0.43
GRU, mean:0.42; median:0.48
SAC-SMA+Snow17 Val, mean:0.66; median:0.71
NSE


Experiment 2 Validation NSE
LSTM, mean:-0.03; median:0.36
GRU, mean:-0.05; median:0.34
SAC-SMA+Snow17 Val, mean:0.66; median:0.71
NSE


Experiment 3 Validation NSE
LSTM, mean:0.44; median:0.5
GRU, mean:0.44; median:0.51
SAC-SMA+Snow17 Val, mean:0.66; median:0.71
NSE