TUM School of Computation, Information and Technology
Department of Computer Science
Technical University of Munich

Applying recurrent neural networks (RNNs) in the field of hydrology to explore uncertainty in time series forecasts and enhance theory-based models

Erik A. Maurer

Thesis for the attainment of the academic degree

Bachelor of Science (B.Sc.)

at the School of Engineering and Design of the Technical University of Munich.

Examiner:
Prof. Dr. Hans-Joachim Bungartz

Supervisor:
Ivana Jovanovic Buha, M.Sc. (hons)

Submitted:
Munich, 21.10.2024

Abstract

Long-short term memory (LSTM) networks perform well in rainfall-runoff modelling or streamflow prediction. Most existing work has concentrated around training networks with meteorological forcings available with CAMELS data. There exist approaches which, in addition to the meteorological forcings, also incorporate static attributes of basins. Further, initial work was done to allow for uncertainty quantification by training Bayesian LSTMs. In this paper, the mentioned approaches are combined and a new architecture to incorporate the static attributes is tested. Catchments from the novel CARAVAN dataset are used to train an LSTM-based encoder-decoder network as found in Zhu and Laptev (2017). Static basin attributes, which are also provided with CARAVAN, are added to the embedding produced by the encoder and then fed through a plain feed forward neural network. Uncertainty quantification is achieved by Monte Carlo Dropout (MCD). Achieving roughly similar loss and accuracy statics, the prediction results from this architecture are broadly in line with existing work on deep learning in the field of hydrology.

Contents

1 Introduction

Machine learning models have the potential to improve the understanding of many different scientific fields (Shen, 2018). One of these fields is hydrology and with it rainfall-runoff predictions. In such predictions, hydrologists attempt to accurately predict the discharge or streamflow. Streamflow, in turn, is the volume of water passing a point in a river over a certain time period (Gauch and Lin 2020). To achieve this goal, meteorological data, also called "forcings," are widely used. These often include temperature, amount of rainfall and ambient pressure measured at different time intervals. As streamflow predictions are both temporal as well as spatial problems (Gauch and Lin 2020), a corresponding area must be defined. This area is referred to as catchment or basin and corresponds to the area from which all precipitation will ultimately run through the designated measurement point at the river. While there have been largely process-based models, which attempt to capture the real world physical processes in mathematical equations and then run simulations, data-driven approaches have emerged in the more recent past (Gauch and Lin 2020). Kratzert et al. (2018) argue that improvements in computing power as well as improved data availability paved the way for these new approaches. A particular class of data-driven models employ a specific recurrent neural network (RNN) architecture which known as a Long Short-Term Memory (LSTM) network. While this architecture was developed by Hochreitner and Schmidhuber (1997), other work such as Gers et al. (2000) further added components to the original architecture. In their advanced form, LSTMs have been shown to be well suited for time series predictions across numerous fields such as, among many others, heat load prediction of buildings (Wang et al 2023), detection of concrete damage (Andrushia et al. 2022), oil production (Panja et al. 2022, Huang et al. 2022), wastewater treatment (Xu et al. 2023), battery temperature prediction (Liu et al. 2024) and prediction of loads for improving performance of computers (Jayanthi and Vallikannu 2022). Kratzert et al. (2018) first demonstrated that the LSTM architecture is also successful in rainfall-runoff or streamflow predictions and even outperformed some of the exisiting process-based models. Others, such as Kratzert et al. (2019), Gauch et al. (2021) and Klotz et al. (2022) further built on this initial work and adopted the basic LSTM architecture to improve prediction accuracy even more. This paper develops another, alternative LSTM-based architecture and discusses its performance in streamflow predictions.

In a first effort, Kratzert et al. (2018) trained out-of-the-box, catchment-specific LSTMs. While these models showed a good performance, Kratzert et al. (2018) also hinted at potential shortfalls. The LSTMs exhibited a rather poor performance in dry or arid basins which are characterized by low values in streamflow. To mitigate this, basins were grouped based on their geographical proximity and new networks were trained on this combined-basins data. These models, however, still performed worse when predicting streamflow for the more arid basins. This led Kratzert et al. (2018) to hypothesize that the low correlation of data across arid basins or regions made it difficult for the model to learn patterns.

To tackle the shortfall of bad predictions in more arid catchments one strategy is to use static catchment attributes along with dynamic forcing data. While Kratzert et al. (2018) used only classical meteorological forcings such as temperature and precipitation on a daily frequency, Kratzert et al. (2019) began using also static catchment attributes such as, among others, an aridity index, the mean elevation of the catchment, etc. These additional, constant variables are passed to the LSTM at every time step as well. Including these attributes helped to improve the model performance. This is intuitive as for example a time series covering a period of heavy rainfall might have very different consequences for streamflow in an arid basin than in a basin where it rains more often. This proved to be especially helpful in the setting of Kratzert et al. (2019) as LSTMs were trained on data stemming from all basins. In addition to working with this out-of-the-box LSTM, Kratzert et al. (2019) experimented with using a different loss function during training. The goal of this change was to make the model more sensitive to low-flow basins. Introducing this adopted loss function indeed improved the performance in such low-flow catchments but had rather

small effects overall. In a third and final approach, Kratzert et al. (2019) alter the standard LSTM architecture and introduce a separate input gate for the static variables. This way, static and dynamic attributes have different routes through the networks. The routes, however, are interdependent such that given a certain value of a static attribute, the LSTM can learn to put different weights on dynamic data. This architectural change was rather complex but also had only limited effects on the final model performance. Another result Kratzert et al. (2019) obtained is that the training procedure is to some degree affected by randomness, e.g. through random weight initialization, and that training entire model ensembles helped to mitigate effects originating from randomness, thereby further enhancing predictions.

Another shortfall of the models of Kratzert et al. (2018) and Kratzert et al. (2019) is that they only provide point estimates. Yet, especially in the field of hydrology, one can easily imagine that uncertainty estimates are useful, e.g. when warning the population of extreme events such as heavy rainfall, flooding, or drought. Uncertainty in predictions originates from multiple sources. Klotz et al. (2022), for instance, distinguish epistemic and aleatoric uncertainty. The first corresponds to the model uncertainty and the second corresponds to the uncertainty inherent to the data. Apart from the requirement of being accurate, models that also report the associated uncertainty must also show good resolution to provide useful predictions (Klotz et al. 2022). This essentially means that uncertainty bands must be reasonable, i.e. neither too specific nor too general, to be actually useful. Klotz et al. (2022) put forward a set of basic statistics to assess model resolution, such as among others mean absolute deviation or variance of predictions.

To allow for uncertainty quantification, Fill (2021) trained catchment-specific models with the Bayes by Backprop algorithm introduced by Blundell et al. (2015). This approach allows for uncertainty quantification as distributions over weights rather than point estimates are derived. One then also obtains distributions over predictions and can sample actual predictions from these distributions. Models trained by Fill (2021) exhibited exceptional accuracy as well as resolution as uncertainty intervals around predictions were reasonably wide. Klotz et al. (2022) also incorporated uncertainty quantification into their streamflow predicition LSTM models. In contrast to Fill (2021), however, they use other approaches such as different variants of mixture density networks and Monte Carlo dropout. Mixture density networks, are trained to return vectors containing the required parameters as well as the relative weights of the distributions considered. For example, if one chooses to combine three Gaussian distributions, the model would simply be trained to put out vectors containing first the three means, then the three standard deviations as those are the two parameters required to specify a Gaussian, as well as a third vector containing the three weights of the distributions. The benefit of this approach is that one can specify any number of components and combine different types of distributions and thereby tailor the model to typical patterns appearing in hydrological data. Monte Carlo dropout, in turn, is yet another technique to quantify uncertainty. Compared to Bayes by Backprop or mixture density networks, this approach is not as complex. One simply does multiple passes through a model but deactivates certain routes with a given probability. This yields varying outputs for the exact same input which, in turn, allows to calculate the statistics Klotz et al. (2022) suggest as resolution benchmarks. Considering their different models, Klotz et al. (2022) found that the Monte Carlo dropout approach yielded the worst overall results in terms of uncertainty estimates. Their hypothesis is that the bad performance of the Monte Carlo method is mostly explained by the fact that the approach assumes a Gaussian distribution of streamflows which, according to Klotz et al. (2022), appears to be violated in times of extremely high or low flows. In fact, they report that especially mixture density networks with asymmetric distributions performed well. However, they regard their MDN models as being overly confident and argue that the MDN models omit epistemic uncertainty. Finally, Klotz et al. (2022) found that uncertainty was lowest around low flows and was elevated during high flows.

Since the introduction of LSTMs to the field of hydrology by Kratzert et al. (2018), the community has worked on the two overarching topics of how to include static catchment attributes and to ivestigate approaches that allow for uncertainty quantification. Coming from a completely different field, Zhu and Laptev (2017) proposed another, alternative LSTM-based architecture which allows to feed in different sets of variables at different stages of the model and also allows to perform reliable uncertainty quantification. While they are concerned with anomaly detection in the ride hailing industry, their so called encoder-decoder approach can also be applied in other domains. This work transfers the model sug-

gested by Zhu and Laptev (2017) to the field of hydrology. The encoder-decoder model features three parts. The first is an encoder which, in a pre-training step is trained in combination with a decoder. While encoder and decoder are two separate LSTMs they are still trained together. The encoder's task is to encode a provided time series into an embedding. The decoder then receives this embedding as well as a truncated sequence consisting out of some of the most recent observation of the time series and produces a prediction by decoding the inputs. This architecture is meant to ensure that the encoder produces an effective embedding that is useful to the decoder. The embedding can be much higher in the dimensions than the actual sequence, thereby giving the possibility to encode much more information in it. After the encoder is trained to produce reasonable embeddings, instead of passing it to the decoder, the embedding is first concatenated with static catchment attributes and then fed through a feed forward network. This last network is trained to produce actual predictions. This is reminiscent of the approach suggested Kratzert et al. (2019) who argued that their architectural changes allow the network to activate certain routes of the dynamic data based on values of static data. In the architecture put forward by Zhu and Laptev (2017), the model is given the ability to asses the exact same embedding differently if it originated from different catchments. The most apparent difference to the Kratzert et al. (2019) approach is that their model receives the static attributes during every single time step of an input sequence. In the design implemented in this paper, the model receives the static attributes at a later stage, only once it has learnt to produce effective embeddings.

The architecture suggested by Zhu and Laptev (2017) also allows for extensive uncertainty quantification. In contrast to Fill (2021) and Klotz et al. (2022), the work presented here does neither use the Bayes by Backprop algorithm nor mixture density networks. Uncertainty quantification is achieved with dropout techniques and is therefore closely related to the Monte Carlo dropout model of Klotz et al. (2022). Further, while Klotz et al. (2022) argue that Monte Carlo dropout in its simple form omits aleatoric uncertainty, Zhu and Laptev (2017) suggest an additional method to estimate the inherent noise of the underlying data. This is achieved by passing observations of a separate set of data through the model and comparing predictions to corresponding ground truth values. Epistemic uncertainty is estimated by variational dropout, a special form of Monte Carlo dropout applicable when working with LSTMs. In terms of uncertainty quantification, the encoder-decoder architecture introduced by Zhu and Laptev (2017) might be considered superior to the architectures discussed by Klotz et al. (2022) as it allows to simultaneously incorporate multiple sources of uncertainty.

Results of the Zhu and Laptev (2017) encoder-decoder architecture are broadly in line with existing work. The best model achieved a median Nash-Sutcliffe-Efficiency (NSE) (Nash and Sutcliffe 1970) of 0.50 across catchments. This is worse than the Kratzert et al. (2019) out-of-the box architecture which achieved a NSE of 0.70 with a single model. This deviation might be caused by a handful extreme outliers from which the model in the paper at hand suffered as the NSE statistics in low performing basins are far worse then the minimums reported by Kratzert et al. (2019). When inspecting correlations between static attributes and achieved model performance the findings of Kratzert et al. (2018) cannot be reproduced with the Zhu and Laptev (2017) architecture implemented here. Basins in which the model performed worst are distributed over a broad range of values of catchment aridity statistics.

In terms of uncertainty, the median normalized standard deviation of predictions was 0.87 across catchments. While Fill (2021) achieved 0.12 and Klotz et al. (2022) achieved 0.13 with the Monte Carlo dropout model, Klotz et al. (2022) also cautioned that their model was too confident, especially around high flow volumes. As the model here captures more sources of uncertainty at the same time, it might be that uncertainty intervals are greater.

While existing literature mostly works with the CAMELS-US dataset (see Addor et al. 2017 or Newman et al. 2022), a novel dataset called CARAVAN was recently introduced by Kratzert et al. (2023). CARAVAN includes many of the catchments also supplied with CAMELS-US along with meteorological forcings as well as static catchment attributes. In this regard, the CARAVAN dataset does not differ in the underlying data that much from CAMELS. A more detailed introduction of the data can be found in section 4. This paper is among the first to build a data-driven surrogate model for streamflow prediction that uses CARAVAN data.

The rest of the paper is structured as follows. Section 2 provides a more detailed introduction to the machine learning theory behind LSTMs and the different ways uncertainty can be incorporated. It also discusses existing literature concerned with deep learning models in hydrology. Section 3 explains the approach by Zhu and Laptev (2017) and how it was adopted to work in the realm of hydrology. After a brief introduction of the novel CARAVAN dataset in section 4, results are presented and discussed in section 5. Section 6 concludes.

2 Theoretical Background

2.1 Long-Short Term Memory Networks

Compared to plain feed forward neural networks, recurrent neural networks (RNNs) also have feedback loops and connections. That means in an RNN, the output of a neuron can also be used as input for a neuron located at an earlier stage of the network. RNNs work with so called hidden states $\mathbf{h_t}$ which store the state of the network at time t. Equation 2.1, taken from Kratzert et al. (2018), shows a standard update of a hidden state $\mathbf{h_t}$ which involves combining an existing, previous hidden state $\mathbf{h_{t-1}}$ with new input data $\mathbf{x_t}$ via linear matrix operations involving weight matrices $\mathbf{W}$ and $\mathbf{U}$, a bias $\mathbf{b}$, as well as a non-linear activation function $g(\cdot)$.

$$\mathbf{h_t} = g(\mathbf{W x_t} + \mathbf{U h_{t-1}} + \mathbf{b}) \tag{2.1}$$

This recurrent nature makes RNNs suited for analyses of time series data and input sequences which are passed in correct temporal order (Rumelhart et al. 1986).

Hochreitner and Schmidhuber (1997) put forward a particular instance of RNNs, so called Long Short-Term Memory Networks (LSTMs). These type of networks perform special operations in certain orders, thereby allowing the network to learn long-term and non-linear patterns in the provided time series. In addition, Hochreitner and Schmidhuber (1997) argue the LSTM operations described below disburden the network from the vanishing or exploding gradient problem, where gradients tend to accumulate and converge towards zero or infinity during training of the network, resulting in poor optimisation behaviour. Following Graves et al. (2013), the involved operations in classical LSTMs can be described by the following seven equations:

$$\mathbf{f_t} = \sigma(\mathbf{W_f x_t} + \mathbf{U_f h_{t-1}} + \mathbf{b_f}) \tag{2.2}$$
$$\tilde{\mathbf{c}}_\mathbf{t} = tanh(\mathbf{W_{\tilde{c}} x_t} + \mathbf{U_{\tilde{c}} h_{t-1}} + \mathbf{b_{\tilde{c}}}) \tag{2.3}$$
$$\mathbf{i_t} = \sigma(\mathbf{W_i x_t} + \mathbf{U_i h_{t-1}} + \mathbf{b_i}) \tag{2.4}$$
$$\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \tilde{\mathbf{c}}_\mathbf{t} \tag{2.5}$$
$$\mathbf{o_t} = \sigma(\mathbf{W_o x_t} + \mathbf{U_o h_{t-1}} + \mathbf{b_o}) \tag{2.6}$$
$$\mathbf{h_t} = tanh(\mathbf{c_t}) \odot \mathbf{o_t} \tag{2.7}$$
$$\hat{y} = \mathbf{W_d h_T} + \mathbf{b_d} \tag{2.8}$$

where vectors $\mathbf{h_{t-1}}$ and $\mathbf{h_t}$ are the hidden states of the network at time steps $t-1$ and t, respectively. Vectors $\mathbf{c_{t-1}}$ and $\mathbf{c_t}$ are the so called cell states. All matrices $\mathbf{W}$ and $\mathbf{U}$ and vectors $\mathbf{b}$ are the weights and biases used in the respective steps or gates. $\mathbf{x}$ is a vector of inputs, $\hat{y}$ is the ultimate output of the network. $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid activation function, which is non-linear, and $\odot$ is used to indicate element-wise multiplication. Fundamentally, the cell state $\mathbf{c_t}$ corresponds to the long-term memory of the network, and the hidden state $\mathbf{h_t}$ constitutes the short-term memory. All equations 2.2 to 2.8 taken together are representative of the inner mechanics of an LSTM cell.

One can then start with equation 2.2 and iteratively plug in equations into one another until one arrives at equation 2.8. The first equation, equation 2.2, represents the forget gate which was first introduced by Gers et al. (2000). As one can see by the multiplication in equation 2.5, the forget gate output is multiplied with the previous cell state $\mathbf{c_{t-1}}$. If some elements in $\mathbf{f_t}$ are small or even zero, parts of $\mathbf{c_{t-1}}$ are lost or "forgotten." Therefore, given the new data $\mathbf{x_t}$ as well as the stored hidden state $\mathbf{h_{t-1}}$, the forget gate $\mathbf{f_t}$ effectively determines which part of the long-term memory $\mathbf{c_{t-1}}$ has become irrelevant. The formation of

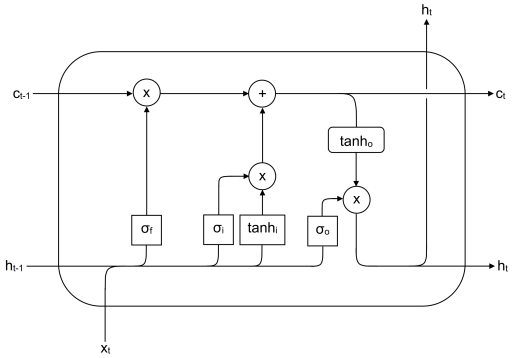


Figure 2.1 LSTM cell. Illustration adopted from Gauch et al. (2021)

a new long-term memory, however, also involves a further step. In equation 2.5 one can also see that $\tilde{\mathbf{c}}_\mathbf{t}$, a new potential cell state, is also required. As in a classical RNN, the calculation of the potential new cell state also involves the past hidden state $\mathbf{h_{t-1}}$ and the input data $\mathbf{x_t}$, as shown in equation 2.3. Instead of adding $\tilde{\mathbf{c}}_\mathbf{t}$ in equation 2.5 directly, it is first multiplied with the output of the input gate $\mathbf{i_t}$. The input gate has quite a similar function as the forget gate, as given the new input data $\mathbf{x_t}$ as well as the stored hidden state $\mathbf{h_{t-1}}$, it determines which parts of the potential new cell state $\tilde{\mathbf{c}}_\mathbf{t}$ contribute to the final new cell state $\mathbf{c_t}$. The update of the hidden state, shown in equation 2.7, involves $\mathbf{c_t}$ as well as the output of a final gate, the output gate $\mathbf{o_t}$. Similar to all the previous gates, given the new input data $\mathbf{x_t}$ as well as the stored hidden state $\mathbf{h_{t-1}}$, the output gate determines which parts of the long-term memory should be transferred also to the short-term memory. At the final stage, a linear feed forward layer, described in equation 2.8, is used to take the current values of the very last hidden layer $\mathbf{h_T}$ and derive a prediction of the time series, denoted $\hat{\mathbf{y}}$. These operations are also illustrated in figure 2.1.

For LSTMs, the temporal order of operations is important. If for example, $\mathbf{x_t}$ is a vector of k dimensions, and the total sequence length is T, one begins to perform the operations described above first with $\mathbf{x_1}$, then continues with $\mathbf{x_2}$ and so on until one arrives at $\mathbf{x_T}$. The LSTM calculates the predicted value(s) for $T + 1$. Note, however, that during this so called "unfolding," the same weights and operations in terms of architecture are used during every step from $t = 1$ until $t = T$. What changes during every step are the values of the involved states $\mathbf{h_{t-1}}$, $\mathbf{h_t}$, $\mathbf{c_{t-1}}$, $\tilde{\mathbf{c}}_\mathbf{t}$, $\mathbf{c_t}$ and the gates $\mathbf{f_t}$, $\mathbf{i_t}$, $\mathbf{o_t}$. The unfolding is also illustrated in figure 2.2. Also note that while the first six LSTM equations presented above, equations 2.2 to 2.7, are

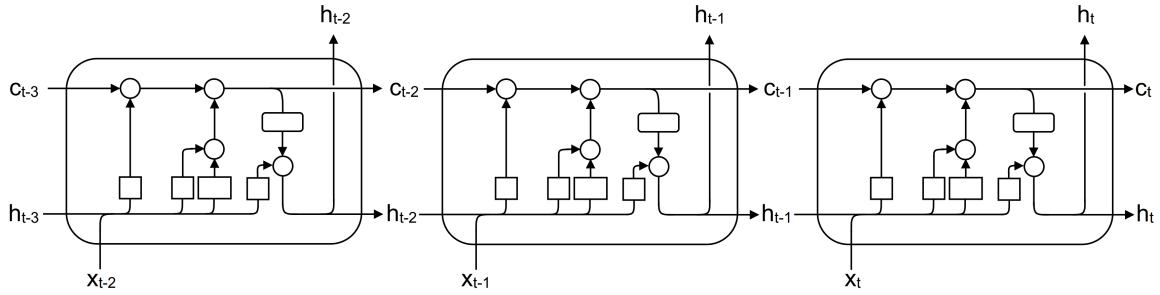


Figure 2.2 Unfolded LSTM cell. Illustration adopted from Kratzert et al. (2018)

evaluated with every new $\mathbf{x_t}$, the final equation, equation 2.8, is only calculated after the very last input for the respective sequence $\mathbf{x_T}$ has been passed through the LSTM.

LSTMs are trained in similar ways as regular neural networks. Training is essentially a repeated optimisation where one tries to adjust weights $\mathbf{W}$ and biases $\mathbf{b}$ to reduce a loss. This loss is calculated by comparing the network's predicted value of the i-th data point $\hat{\mathbf{y}}_\mathbf{i}$ to the corresponding ground truth value $\mathbf{y_i}$

across all n samples. In a regression setting, which involves numerical data, the loss function is usually a function like the mean squared error (MSE), described in equation 2.9.

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

Reducing the loss is only possible if weights and biases of the model can be adjusted such that the value of the loss function decreases. To do so, a gradient w.r.t. the weights and biases is calculated with a method called backpropagation. Given the gradient, weights are then adjusted by a small move towards the direction of the optimum. Exactly by which magnitude weights and biases are adjusted depends on the learning rate, a hyperparameter. Also, as the derivatives depend on the current values of weights, biases and inputs, the derivatives must be calculated with the data provided. Often this is impossible with large datasets and therefore "batches" of data are sampled to use in calculation of the gradient. Such an "optimizer" is known as stochastic gradient descent, but there exist other, more advanced optimization algorithms that are of adaptive nature such as Adam (Kingma and Ba 2017) or AdaGrad (Duchi et al. 2011), etc. A basic introduction to these different techniques can be found in Goodfellow et al. (2016). While the weights and biases are updated towards optimal values during the training procedure, the chosen learning rate, the optimizer, as well as the loss function are hyperparameters and must be chosen by the developer. A common practice is to split the data into training, validation and test samples and train the network with the training data only. Yet the overall goal during training is to reduce the same loss function calculated on the validation set of samples, i.e. some samples the network has not seen during training. This is done to prevent overfitting, which occurs if the model learns too many specific details which are only prevalent in the training data instead of capturing general patterns. Hence to supervise the performance of the model on unseen data one trains with the training data but one checks with help of the validation data.

2.2 Incorporating Uncertainty

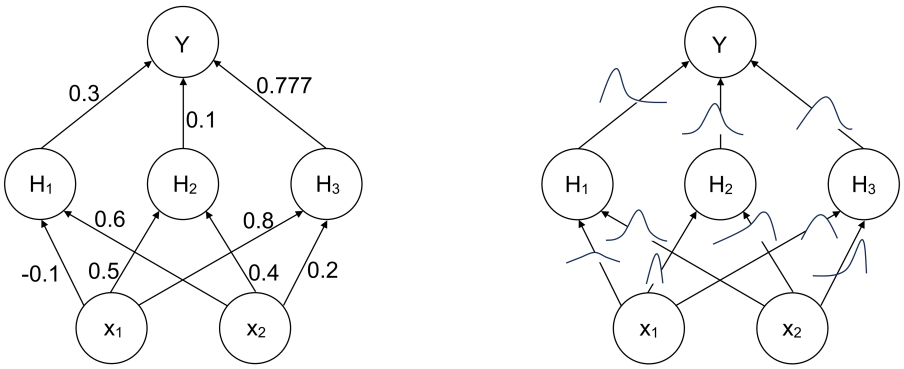
Standard (R)NNs and LSTMs usually provide point estimates only. Given an input sample, a forward pass through the network is made and a single output value is predicted. For a given model prediction, one does not know, how confident the model is in that value. This is a significant disadvantage, especially in real-world environmental applications like hydrology. One does not get an uncertainty estimate in form of e.g. a confidence interval. Prediction uncertainty, however, stems from multiple sources. Zhu and Laptev (2017), for example, distinguish three different types of uncertainties. The first type is model uncertainty which refers to uncertainty resulting from model parameters. It is also known as epistemic uncertainty (e.g. Klotz et al. 2022). If the model parameters are changed slightly, the output of the network can change, yielding also different predictions.

A second type of uncertainty mentioned by Zhu and Laptev (2017) is the uncertainty inherent in the underlying data. For example, if data stems from a normally distributed distribution with variance σ^2, a standard network would still only give a point estimate. Even if the model is perfectly confident in its parameters, meaning low model uncertainty, this second type of uncertainty remains. A provided point estimate always conceals the fact that there is inherent insecurity stemming from the original, underlying data generation process. This type of uncertainty is also referred to as aleatoric uncertainty (e.g. Klotz et al. 2022) and it cannot be reduced.

The third type of uncertainty Zhu and Laptev (2017) discuss is uncertainty originating from model (mis)specification. As Zhu and Laptev (2017) are concerned with anomaly detection, this third type of uncertainty is especially apparent in their setting. During irregular events in time series, a given model might not be able to fit to the new circumstances. This might also be the case in a hydrological setting, where during extreme events such as flood or drought, the entire distribution of the data generating process possibly changes and hence the model which was trained mostly on standard events might not be suited any longer.

2.2.1 Bayes by Backprop

There exist different approaches to incorporate uncertainty estimations into machine learning models. Blundell et al. (2013), for example, put forward an algorithm they named Bayes by Backprop (BBB). The main goal is to derive an entire probability distribution over the weights $\mathbf{w}$ given the data $\mathbf{D}$: $P(\mathbf{w}|\mathbf{D})$. This idea is also illustrated in figure 2.3 where panel (a) shows an example of a standard neural net where all weights are simple scalars. Panel (b) shows a Bayesian neural network where entire distributions over the weights are calculated.



(a) Standard Neural Network trained with standard backpropagation

(b) Bayesian Neural Network trained with BBB

Figure 2.3 Different types of neural networks. Figure adopted from Blundell et al. (2015)

BBB requires a special loss function $F(\mathbf{D}, \theta)$, called the variational free energy or sometimes expected lower bound (ELBO) loss, which can be approximated as follows:

$$F(\mathbf{D}|\theta) \approx \sum_{i=1}^{n} log\ q(\mathbf{w^{(i)}}|\theta) - log\ P(\mathbf{w^{(i)}}) - log\ P(\mathbf{D}|\mathbf{w^{(i)}}) \tag{2.10}$$

In equation 2.10, $q(\mathbf{w^{(i)}}|\theta)$ is the variational posterior and θ is the parameter space of variational parameters consisting out of mean μ and standard deviation ρ such that $\theta = (\mu, \rho)$. Intuitively, what this loss function does is comparing the variational posterior distribution to the ground truth distribution $P(\mathbf{w}|\mathbf{D})$ by means of the Kullback-Leibler (KL) divergence. $P(\mathbf{w^{(i)}})$ in equation 2.10 is a prior over the weights. This can be chosen freely but Blundell et al. (2015) suggest using a scale mixture of two Gaussian distributions:

$$P(\mathbf{w}) = \prod_{j} \pi N(\mathbf{w_j}|0, \sigma_1^2) + (1-\pi) N(\mathbf{w_j}|0, \sigma_2^2) \tag{2.11}$$

The BBB algorithm then proceeds as follows. At each step, it first samples white noise ϵ from multi-dimensional Gaussian distribution $\epsilon \sim N(0, \mathbf{I})$. Then it takes stored values μ and ρ and calculates the weight as follows: $\mathbf{w} = \mu + log(1 + exp(\rho)) \circ \epsilon$. Next, the loss function is calculated as described in equation 2.10 as $f(\mathbf{w}, \mathbf{D}) = log\ q(\mathbf{w}|\theta) - log\ P(\mathbf{w})P(\mathbf{D}|\mathbf{w})$. Then, gradients Δ_μ and Δ_ρ are calculated and the variational parameters are updated:

$$\mu \leftarrow \mu - \alpha\ \Delta_\mu$$
$$\rho \leftarrow \rho - \alpha\ \Delta_\rho$$

The prior $P(\mathbf{w})$ does not change during any optimisation step.

Fortunato et al. (2017) built on the work from Blundell et al. (2015) and extended the BBB algorithm to RNNs. Recalling again how RNNs and LSTMs are set up from section 2.1, one essential difference is that weights are shared between different time instances. Therefore, when a gradient is calculated w.r.t. these shared weights, it will consist out of more parts, namely one part for each time step. Also, especially with long sequences, one must think about when exactly to calculate losses and how to adopt the loss function

to reflect the fact that only minibatches and truncated sequences are used. Fortunato et al. (2017) suggest distributing the KL cost among minibatches and truncated squences and to proceed as follows: For each minibatch of truncated sequences, sample a set of weights with the help of white noise as already done in Blundell et al. (2015). After forward and backward propagation, calculate all necessary gradients (weights and distributional parameters). Then update the distributional parameters.

Apart from transferring the BBB algorithm to RNNs, Fortunato et al. (2017) also introduced posterior sharpening. Instead of using the variational posterior $q(\mathbf{w})$, this algorithm uses a conditional distribution and conditions on the minibatch data: $q(\mathbf{w}|(\mathbf{x}, y))$. This can be achieved efficiently by introducing a latent variable φ, which is then integrated out again:

$$q(\mathbf{w}|(\mathbf{x}, y)) = \int_{\varphi} q(\mathbf{w}|\varphi, (\mathbf{x}, y))q(\varphi)d\varphi$$

with vectors μ, σ, and $q(\varphi) = N(\varphi|\mu, \sigma)$. This then gives $q(\mathbf{w}|(\mathbf{x}, y)) = N(\mathbf{w}|\varphi - \eta \circ g_{\varphi}, \sigma_0^2\mathbf{I})$ where η is an additional parameter the network must learn and which Fortunato et al. (2017) interpret as a per-parameter learning rate, σ_0^2 is a hyperparameter and g_{φ} is the gradient of the loss function w.r.t. to φ.

Zhu and Laptev (2017) discuss the BBB approach and mention drawbacks, including the change of the loss function, the doubling of the parameters needed to estimate as well as rather complex changes to training procedures.

2.2.2 Monte Carlo dropout

A method that is much more straightforward in its implementation, neither requiring changes to model architecture nor to training procedures, is Monte Carlo Dropout (MCD). While MCD was a method to perform weight regularization and prevent overfitting (Srivastava et al. 2014) originally, Gal and Ghahramani (2016) demonstrated that one can also use MCD to quantify uncertainty. In standard NNs, the dropout algorithm randomly selects weights and then sets them to zero, i.e. they are dropped out, hence the name. This means that during every pass, a slightly different network architecture is used. During training, this approach helps to prevent overfitting as the network cannot rely too heavily on a specific network connection. Yet during inference, one can also set a dropout probability and pass the same input multiple times. Each time, different network routes are active, yielding slightly different outputs. This corresponds to a sampling method for Bayesian inference and hence the name Monte Carlo Dropout (Klotz et al. 2022, Gal and Ghahramani 2016). With this method one can obtain multiple predictions for a single sample and quantify uncertainty with statistics such as variance, interquantile distances, etc.

In the context of LSTMs, there is one more detail related to dropout. As described in section 2.1, LSTMs are unfolded over time steps. If one decides to apply dropout, one must decide whether to drop out the exact same weight during every time step or unfolding, or whether at each step the dropout is applied from scratch meaning every weight has the same probability of being dropped out. The former variant, were the exact same weight is deactivated for the entire sequence is known as variational dropout (Gal and Ghahramani 2016) and is used by Zhu and Laptev (2017).

2.2.3 Mixture density networks

Mixture density networks (MDN) as introduced by Bishop (1994) combine multiple probability distributions, called components. These are then combined via a weighted mean. The components can be chosen freely, e.g. one can chose to combine three Gaussian distributions with two Laplacians. The weights then correspond to the probability of each component. As described by Klotz et al. (2022), the MDN approach can be implemented by designing a network not to return a direct prediction but the respective component weights together with parameters of the chosen distributions. In the above example, instead of returning a point estimate, the model would return a vector with means for Gaussians and Laplacians, a second vector with scale parameters of the two distributions and a third vector with weights of the respective distributions. All this information can be used to derive a point estimate by sampling from the obtained mixture distribution, along with uncertainty bands or other statistics.

2.3 LSTMs in Hydrology

While LSTMs concerned with numerical data are often trained with the MSE loss (see section 2.1), the quality of time series predictions in hydrology is usually measured by different metrics. The most popular is the Nash–Sutcliffe-Efficiency (NSE), introduced by Nash and Sutcliffe (1970). It is calculated as described in equation 2.12 where T is the total number of time steps, $Q_{\text{obs},t}$ is the observed value of the time series at time step t, $Q_{\text{pred},t}$ is the predicted value at time step t and $\overline{Q_{\text{obs}}}$ is the mean of the observed values over all time steps.

$$\text{NSE} = 1 - \frac{\sum_{t=1}^{T} \left(Q_{\text{obs},t} - Q_{\text{pred},t}\right)^2}{\sum_{t=1}^{T} \left(Q_{\text{obs},t} - \overline{Q_{\text{obs}}}\right)^2} \tag{2.12}$$

The NSE ranges from $-\infty$ to 1.00, where 1.00 indicates a perfect prediction. Newman et al. (2015) put forward an NSE of greater than 0.80 as an indicator for a well-performing model. In hydrology, the variable of interest, Q, is usually the streamflow measured in cubic meters per second. Many models are concerned with predicting the streamflow based on meteorological forcings such as mean precipitation, temperature, pressure, etc. This prediction is often called rainfall-runoff predicition or, alternatively, streamflow prediciton (Gauch and Lin 2020). For this, a given land area is split into so called basins or catchments. The idea of a basin is that all rainfall will eventually pass a certain point (Gauch and Lin 2020). This point is equipped with a gauging station which is used to measure how much streamflow is running through.

Long-Short Term Memory (LSTM) networks have been proven to be capable of handling hydrological processes an provide useful rainfall-runoff predictions (Kratzert et al. 2018, Kratzert et al. 2019, Kratzert et al. 2020, Gauch et al. 2021, Nevo et al. 2022 and Klotz et al. 2022). While many approaches use LSTMs in principle, many different LSTM variants and architecture configurations have been applied.

Kratzert et al. (2018) first applied LSTMs in the field of hydrology and mostly focused on training basin-specific networks for 241 CAMELS US (see Addor et al. 2017, Newman et al. 2022 for a detailed description of the dataset) catchments spanning four different hydrological units (HUCs). An LSTM architecture with two stacked and 20 hidden layers was chosen. During training, Kratzert et al. (2018) evaluated the model performance against the MSE and used a batch size of 512. Sequences of length 365 were passed and a dropout probability of ten percent was used. As a basic result of these basin-specific LSTMs, Kratzert et al. (2018) found that the LSTMs in general either outperformed or matched the performance of the benchmark, the process-based Soil Moisture Accounting Model + Snow17 (SC-SMA + Snow17). On average, the catchment specific models achieved a NSE of 0.63. Apart from the main result, Kratzert et al. (2018) also found LSTMs underestimated peaks more strongly compared to the process-based benchmark. Another potentially relevant result is that LSTMs perform worse in dry, arid basins where there is a streamflow of zero for a high number of consecutive days.

2.3.1 Regionalization

Regionalization, in the context of hydrology, refers to training a single model for multiple catchments. Kratzert et al. (2018) performed first regionalization attempts by, in addition to basin-specific networks, also training HUC-specific models. HUCs are groups of catchments, where the grouping is based on geographical properties. For example, many catchments are grouped together as they lie next to the same river. Results from these regional models showed a mixed picture. Across all basins, the HUC-specific models of Kratzert et al. (2018) showed little differences in results compared to basin-specific ones. The variance, however, increased as the respective regional models performed better in almost all basins of the New-England HUC but it underperformed basin-specific models in about 20 percent of all catchments. One potential reason put forward by Kratzert et al. (2018) might be that streamflows are more correlated in the New England HUC than in other HUCs. Kratzert et al. (2018) therefore hypothesized, that regional LSTMs might be far better in predicting streamflows across multiple basins if the streamflows are highly correlated. Kratzert et al. (2018) further found that while basin-specific LSTMs rather underestimated the actual streamflow, regional models showed more balanced over- and underestimations.

In a final variation of their approach, Kratzert et al. (2018) continued training regional models with basin-specific data only. This strategy tries to first make the model learn the general patterns across basins before learning the nuances of processes governing the discharge in the respective basin. This fine tuned hybrid variant again improved the model accuracy compared to both purely basin-specific and purely HUC-specific models. With this last strategy a mean NSE of 0.68 across all 241 basins was achieved.

In another study, Kratzert et al. (2019) also performed regionalization and trained three different variants of LSTMs on 531 CAMELS US catchments. While they also experimented with a different loss function, they used the MSE in all the models for comparison. After a grid search over possible parameters of the LSTM implementations, they found a single stacked LSTM layer with 256 hidden states and a sequence length of 270 to work best. During training, a dropout rate of 40 percent was used to prevent overfitting.

The first model architecture was a simple, out-of-the-box LSTM which was trained on input sequences of all basins and achieved an average NSE across basins of about 0.36 with 44 basins having a negative NSE.

Kratzert et al. (2019) extended their approach and enriched their data with static basin attributes, such as, among many others, mean elevation and forest fraction. This static data is constant across all sequences of a given basin. Passing these attributes along the dynamic, time-varying data helped to lift the average NSE to 0.71 and lowered the number of basins where the NSE turned out to be negative to only six basins.

Finally, Kratzert et al. (2019) manipulated the internal architecture of an LSTM cell to allow dynamic and static inputs to flow through different input gates of the cell. The idea behind this approach was to enable the network to selectively enable and disable certain parts of the flow of dynamic inputs based on the respective static inputs. On the one hand, the model can learn to activate the same LSTM parts for hydrologically similar basins and on the the other hand, it can also learn to tell apart the exact same meteorological forcings occurring in different basins. Additionally, it can learn to incorporate information or processes that are equal across both all basins as well as similar basins. Similarity, in the case of this differently-configured LSTM, can be thought of as an embedding. Compared to the out-of-the-box architecture fed with both dynamic and static inputs, however, this architectural change slightly decreased the basin-averaged NSE to 0.68 and yielded a number of basins with negative NSE of six.

Kratzert et al. (2019) also benchmark their models against process-based models such as the VIC (Liang et al.1994) as well as the mHM (Samaniego et al. 2010, Kumar et al. 2013). These two models were also regionally calibrated on data from 447 basins. Even the out-of-the box LSTM without the static attributes outperformed the two process-based models as it achieved an average NSE of 0.71 across these 447 basins. The VIC, in contrast, achieved an NSE of 0.17 and the mHM an NSE of 0.44. Both process-based models outperformed the LSTM in less than five percent of the basins.

Concerning the relevance of the static features, Kratzert et al. (2019) used the method of Morris (Morris 1991) to provide a feature ranking of relevance. This method uses the fact that neural networks are functions that are differentiable. Thus one can calculate the derivatives w.r.t. specific input features and use the results to rank the input features across their relevance. Kratzert et al. (2019) found that vegetational indices as well as soil features are ranked lowest. In contrast, climate indices such as aridity or mean precipitation and topological indices like catchment area or mean catchment elevation are ranked high. This is in line with most of the results obtained by others such as Addor et al. (2018).

2.3.2 Uncertainty-aware models

As discussed in subsection 2.2, there exist implementations of LSTMs that allow for quantification of uncertainty in obtained predictions. These approaches have also been applied in the hydrological domain where uncertainty quantification is particularly relevant and useful.

Klotz et al. (2022) trained LSTMs which are fed both static catchment attributes and meteorological forcings. Three MDNs as described in section 2.2.3 and a MCD model as described in section 2.2.2. All models were trained with basins provided with the CAMELS US dataset. The training loss for the MCD model was also the MSE, the hidden size of the LSTM was 500. The MDN models were trained with 250 hidden units. All models were trained for 30 epochs on data originating from all 531 CAMELS basins. The

MCD model trained by Klotz et al. (2022) achieved a basin-average NSE of 0.65 and therefore slightly underperformed the MDN models which achieved average NSEs between 0.69 and 0.75.

Fill (2021) used an implementation by Esposito (2020) to train basin-specific bayesian LSTMs using the BBB algorithm of Blundell et al. (2015) and Fortunato et al. (2017). Three different LSTM variants, that differ w.r.t. the prediction horizon as well as whether past streamflows are included or not, were implemented. Models were trained with two stacked layers, each with 50 hidden units for 50 epochs. The most successful model was a model that predicted one time step into the future and got past discharges as an input. It achieved an average NSE of 0.81. Compared to most of the studies mentioned above (Kratzert et al. 2018, Kratzert et al. 2019, Klotz et al. 2022) that study trained basin-specific models and therefore might be superior in predicting streamflows.

2.3.3 Training LSTMs in a hydrology context

Kratzert et al. (2018) trained an LSTM with 20 hidden units and noted that especially for LSTMs with a low number of hidden units, initialization of weights matters. They tackle this problem by training ensemble models. In this approach, multiple architecturally equivalent LSTMs are trained and for the final prediction the average of all outputs is used. Kratzert et al. (2019) also train ensembles to account for the initialization error. They found that using ensembles in predictions lowered the number of basins for which the NSE was below zero.

Kratzert et al. (2019) train models on 531 catchments. For this, they introduce a basin-averaged NSE as new loss function. They argue this is necessary because if data of multiple basins is taken into account, the linear relationship between the MSE loss function (specified in equation 2.9) and the NSE (specified in equation 2.12) breaks down. This happens because the NSE is a normalized variant of the MSE and if data originates from more than one basin, the normalization constant changes. A direct consequence is that when regional models are trained with an MSE loss, basins with low average discharges are automatically less relevant during training as the model already assumes it performs reasonably in those basins, resulting in lower gradient adjustments and ultimately in a low model performance in these basins when evaluated against the NSE. In their work, Kratzert et al. (2019) found that using the basin-averaged NSE as a loss function compared to the MSE indeed improved the performance in basins where the NSE statistic was low. The effect, however was of small magnitude when LSTMs which also considered static catchment attributes were considered. For example, the simple out-of-the-box LSTM which received both meteorological forcings and static attributes achieved a mean NSE of 0.71 across basins when trained with the MSE loss and a mean NSE of 0.72 when trained with the basin-averaged NSE as a loss function.

3 Methodology

3.1 Encoder-Decoder Model

In the context of ride sharing, Zhu and Laptev (2017) used an LSTM-based encoder-decoder model to predict the number of trips for the American ride-hailing services company *Uber Technologies, Inc* in a handful American mega cities. The benefit of the encoder-decoder architecture, which is explained in more detail below, is that it allows for extensive uncertainty quantification in predictions. As outlined in section 2.2 above, Zhu and Laptev (2017) mention three different components of uncertainty: First, model uncertainty which is sometimes referred to as epistemic uncertainty and second, inherent noise in the data, sometimes referred to as aleatoric uncertainty. The main advantage of their approach is that it also takes into account the third type of uncertainty, which results from model miss-specification. The goal of this work is to implement a similar strategy in the hydrological realm as uncertainty quantification is also of crucial importance in hydrological predictions. Zhu and Laptev (2017) are particularly concerned with anomaly detection. In their setting, regime changes in the data generation process might occur on special days such as Christmas Eve, New Year or Labor Day during which the ride hailing services company often experiences spikes in the number of requested rides. Zhu and Laptev (2017) argue that during such events, data might be generated from a different distribution and a model fitted on regular days might result in systematically wrong predictions. Their model, takes into account this model miss-specification. Extreme events also occur in the hydrological setting. One can easily imagine times of extreme precipitation such as Hurricanes or severe droughts. Thus, the approach might also be superior in the field of hydrology.

3.1.1 Model parts

There are three main components of the Zhu and Laptev (2017) encoder-decoder model: 1. the encoder, 2. the decoder and 3. a simple feed forward neural network. This architecture implements the following logic. Every input sequence $\{\mathbf{x_t}\}_t = \mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_{T-1}}, \mathbf{x_T}$ of length T is first passed through the encoder LSTM. After passing the last day of the input sequence, the hidden and cell states of the encoder LSTM are passed to the decoder. The decoder uses these passed states to initialize another, new LSTM. The dimension of these passed states depend on the chosen sizes of the LSTMs and are fixed. They are also named the *embedding* of the input sequence. The newly initialized LSTM is trained to predict the actual target day(s), $\mathbf{x_{T+1}}, \mathbf{x_{T+2}}, ..., \mathbf{x_{T+F-1}}, \mathbf{x_{T+F}}$. The decoder LSTM takes as an input also parts of the sequence $\{\mathbf{x_t}\}$, namely $\mathbf{x_{T-F+1}}, \mathbf{x_{T-F}}, ..., \mathbf{x_{T-1}}, \mathbf{x_T}$. In the field of hydrology it is common to predict the streamflow of the next day. This approach was also taken here, so in this case $F = 1$. In other words, the output sequence has a length of one. Then, the decoder LSTM, with the help of the embedding, predicts y_{T+1} based on the input $\mathbf{x_T}$. The architecture is also illustrated in figure 3.1. This model architecture is meant to guarantee that the encoder generates a useful embedding of the provided sequence. In other fields such as natural language processing, such encoder-decoder models are also referred to as sequence-to-sequence (seq-2-seq) models and the embedding is often referred to as the context vector (Sutskever et al. 2014). This perspective also gives a nice interpretation in the field of hydrology. Taken for itself, it is quite hard to predict the next day's streamflow from the last day's meteorological forcings. Yet when the decoder also receives the context, meaning an encoded version of the past T days' forcings, it should make it easier to predict the streamflow.

Although this follows standard implementation of an encoder-decoder, Zhu and Laptev (2017) propose to use the embedding in a different way. Once the encoder-decoder setup is trained in a first "pre-training"

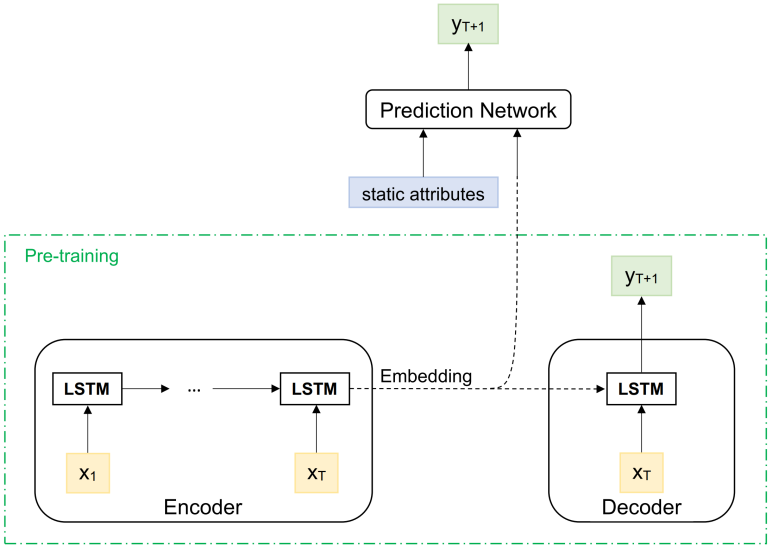


Figure 3.1 Encoder Decoder Approach. Figure adapted from Zhu and Laptev (2017).

step, they use the trained encoder to create embeddings of training sequences and then pair the embeddings with the respective external features. The external features, in their case, are factors like city population growth. In the field of hydrology, the static attributes are used. This concatenated input of embedding plus static catchment attributes is then fed through a standard feed forward network which is trained to make the actual prediction of the next day's streamflow. The entire approach is illustrated in figure 3.1.

3.1.2 Uncertainty estimation

In this framework, uncertainty can be estimated using Monte Carlo Dropout techniques described in section 2.2.2. A neural network can be described by a mathematical function $f^{\mathbf{W}}(\cdot)$ where f represents the chosen architecture and $\mathbf{W}$ is the entire parameter set of weights and biases. As described in more detail above, Bayesian neural networks start with a prior on the parameters $P(\mathbf{W})$ and, given the data $\mathbf{D}$, try to find an optimal posterior distribution $P(\mathbf{W}|\mathbf{D})$. Following Zhu and Laptev (2017), given a new datapoint $\mathbf{x}^*$, the predictive distribution can be obtained by

$$p(y^*|\mathbf{x}^*) = \int_W P(y^*|f^{\mathbf{W}}(\mathbf{x}^*))\ P(\mathbf{W}|\mathbf{D})\ d\mathbf{W} \tag{3.1}$$

Given this predictive distribution, the total prediction uncertainty can be obtained from the law of total variance:

$$\text{Var}(y^* \mid \mathbf{x}^*) = \text{Var}\left[\mathbb{E}(y^* \mid \mathbf{W}, \mathbf{x}^*)\right] + \mathbb{E}\left[\text{Var}(y^* \mid \mathbf{W}, \mathbf{x}^*)\right] \tag{3.2}$$

$$= \text{Var}(f^{\mathbf{W}}(\mathbf{x}^*)) + \sigma^2 \tag{3.3}$$

The first term, $\text{Var}(f^{\mathbf{W}}(\mathbf{x}^*))$ captures the uncertainty stemming from uncertainty from model parameters $\mathbf{W}$. Zhu and Laptev (2017) argue that this part truly encompasses two sources of uncertainty: uncertainty stemming from model parameter values (epistemic uncertainty) as well as uncertainty from model miss-specification (wrong architecture $f(\cdot)$). The second term, σ^2, captures the third type of uncertainty, namely the uncertainty inherent to the data generating process (aleatoric uncertainty). This third type cannot be minimized, even with a perfectly specified and trained model.

The first two types of uncertainty can be quantified with the help of MCD. A given input sample, denoted $\mathbf{x_i}$, is passed first through the encoder. The obtained embedding is then passed through the feed-forward network. During these passes, variational dropout is activated in the encoder stage. In the feed forward network stage, dropout is activated, too. With dropouts, if the same $\mathbf{x_i}$ is passed through the

network, one obtains different outputs. Assume the entire procedure is repeated R times. Then one obtaines $\{\hat{y}_1, \hat{y}_2, ..., \hat{y}_{R-1}, \hat{y}_R\}$. These values can then be used to estimate the uncertainties from model miss-specification and the uncertainty stemming from model uncertainty w.r.t. the involved parameters, both captured in Var($f^{\mathbf{W}}(\mathbf{x_i})$):

$$\text{Var}\left(f^{\mathbf{W}}(\mathbf{x_i})\right) = \frac{1}{R}\sum_{r=1}^{R}(\hat{y}_r - \bar{\hat{y}})^2 \tag{3.4}$$

where $\bar{\hat{y}} = \frac{1}{R}\sum_{r=1}^{R}\hat{y}_r$ is the sample mean. The uncertainty from model miss-specification is captured by the encoder-decoder approach. During training, the encoder learns to construct a reasonable embedding of provided training samples. If, during inference, an unusual sample is passed, the embedding should vary more because the encoder has never seen such a sample. This is particularly useful in the case of anomaly detection as in Zhu and Laptev (2017) but also in the context of hydrology and extreme events like flooding or drought.

Special to the approach of Zhu and Laptev (2017) is that they estimate the inherent uncertainty, σ^2, from an independent held-out validation set of samples $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_{V-1}}, \mathbf{x_V}\}$, $\mathbf{Y} = \{y_1, y_2, ..., y_{V-1}, y_V\}$. If $f^{\hat{\mathbf{W}}}(\cdot)$ is the trained model, then σ^2 can be approximated by

$$\sigma^2 = \frac{1}{V}\sum_{v=1}^{V}\left(\hat{y}_v - f^{\hat{\mathbf{W}}}(\mathbf{x_v})\right)^2 \tag{3.5}$$

3.1.3 Inference

To come up with a point estimate as well as an uncertainty measure one performs the algorithm described in algorithm 1. It requires a trained encoder $e(\cdot)$, a trained feed forward network $h(\cdot)$ an input sequence $\{\mathbf{x_t}\}_t$ with the corresponding static catchment attributes ζ from the basin the sequence originates from and two hyperparameters, namely the dropout probability p and the number of Monte Carlo draws R. A respective sequence is first passed through the encoder to obtain an embedding. The embedding is concatenated with the static input features and both are passed through the feed forward prediction network. This procedure is repeated R times such that R predictions $\hat{y}_r$ are obtained. As a dropout probability p is active during this inference forward pass, the predictions vary slightly and the final prediction for the sequence, $\hat{y}_{t+1}$, is obtained by calculating the average of all predictions $\hat{y}_r$.
The uncertainty of $\hat{y}_{t+1}$ is a combination of two components. The first component η_1^2 can be obtained by calculating the variance of the R predictions. The second component, η_2^2, is obtained independently, from the validation set. For this uncertainty component, predictions are derived for the validation period sequences and they are compared to the ground truth values. Finally, algorithm 1 returns the point estimate as well as the standard deviation.

3.1.4 Model implementations

Starting from scratch, the very first implementation of the encoder-decoder approach followed the architecture from Fill (2021). This included choosing only one stacked layer, 50 hidden units, and a sequence length of 30 days. Then, after some experiments with both simpler as well as more complex model architectures, the hyperparameters from Kratzert et al. (2019) were investigated. This was considered to be reasonable as Fill (2021) trained basin-specific networks but Kratzert et al. (2019) performed regionalization by training their LSTMs on data from multiple basins. Their setup was considerably more complex compared to Fill (2021) as Kratzert et al. (2019) used 256 hidden units and a sequence length of 269 days to predict the 270th day. Playing around with the network complexity as a form of informal hyperparameter search, architectures between the Fill (2021) and Kratzert et al. (2019) settings were chosen. Klotz et al. (2022) even chose a higher number of hidden units yet this was not implemented as early results were rather seen as indicative of the model being too complex rather than being too simple. This also ruled out the settings from Kratzert et al. (2018) of two stacked layers, 20 hidden layers and sequence lengths of 365. All trained encoder-decoder models had a single stacked layer. Hyperparameter search in

Algorithm 1 Inference. Adopted from Zhu and Laptev (2017)

Inputs: trained encoder $e(\cdot)$, trained feed forward network $h(\cdot)$, dropout probability p, number of Monte Carlo iterations R, input sequence $\{\mathbf{x_t}\}_t$, static catchment attributes ζ, validation set of V observations $\{\mathbf{x_1}, ..., \mathbf{x_V}\}$ with corresponding targets $\{y_1, ..., y_V\}$

Result: prediction $\hat{y}_{t+1}$, predictive uncertainty η

Procedure:

for $r = 1$ ***to*** R **do**

$g_r \leftarrow$ Variational Dropout$(e(\mathbf{x_t}), p)$

$z_r \leftarrow$ Concatenate(g_r, ζ)

$\hat{y}_r \leftarrow$ Dropout$(h(z_r), p)$

end

// prediction

$\hat{y}_{t+1} \leftarrow \frac{1}{R} \sum_{r=1}^{R} \hat{y}_r$

// model uncertainty and miss-specification

$\eta_1^2 \leftarrow \frac{1}{R} \sum_{r=1}^{R} (\hat{y}_r - \hat{y}_{t+1})^2$

// inherent noise

for $\mathbf{x_v}$ *in validation set* $\{\mathbf{x_1}, ..., \mathbf{x_V}\}$ **do**

$\hat{y}_V \leftarrow h(e(\mathbf{x_v}))$

end

$\eta_2^2 \leftarrow \frac{1}{V} \sum_{v=1}^{V} (\hat{y}_v - y_v)^2$

// total prediction uncertainty

$\eta \leftarrow \sqrt{\eta_1^2 + \eta_2^2}$

return $\hat{y}_{t+1}, \eta$

the works mentioned showed that adding a second stacked layer is not that helpful. Concerning the other hyperparameters, Klotz et al. (2022) and Fill (2021) were followed by using a batch size of 256 and similar to Klotz et al. (2022) a learning rate of 0.001 was chosen. Adam was chosen as an optimizer.

Zhu and Laptev (2017) mention that they worked with variational dropout in the encoder-decoder. As discussed in section 2.2.2, variational dropout is a variant of MCD, adopted for RNNs. By relying on an existing implementation by Viviano (2021), this variant was also chosen here.

Different to existing work in the field, one of the approaches considered included changes in streamflow rather than predicting actual levels. Separate encoder-decoder networks were trained on changes and on levels. Yet this route also showed daunting early results and was not further explored.

The pre-training step involves training the encoder-decoder network only. As is described in algorithm 1, the encoder produces an embedding, which is then concatenated with the static basin features. This concatenated vector is then used as training data for the feed forward prediction network. For this prediction network, multiple architectures were considered as well. Here, the very first attempt also involved a complex architecture of four hidden layers with 256, 128, 64 and 32 neurons, respectively. The complexity was then decreased step by step. Next, only three hidden layers with 128, 64 and 32 neurons were considered. After that, models with two hidden layers were implemented. First, one with 64 and 32 neurons, then with 32 and 16, then with 16 and 8, then with 8 and 4 until ultimately a model with as few as 4 and 2 neurons was implemented. Table 5.1 provides an overview of the feed forward architecture iterations that were considered. Brief experiments with using dropout as a regularization technique were also carried out.

3.2 Soft- and Hardware

All software that was used is open-source. Python 3.8.10 (van Rossum 1995) was chosen as a programming language, PyTorch (Paszke et al. 2019) was the library used to implement the deep learning models. While mostly the standard neural network classes were used, changes to implement variational dropout were required in case of the LSTM class. However, these changes were mostly taken from an existing

implementation by Viviano (2021). Preprocessing of data was carried out with Pandas (McKinney 2010), Numpy (Harris et al. 2020) and sci-kit learn (Pedregosa et al. 2011). All charts were created with Matplotlib (Hunter 2007).

Computations were carried out on a high-performance server with a single-socket AMD EPYC 7402 server-class CPU at 2.80 GHz clock frequency. Total RAM was 256 GiB. The available GPU consisted out of four high-end NVIDIA RTX 3080 GPUs, each having 10 GiB of video memory (VRAM). The SSD storage, was connected via Non-Volatile Memory Express (NVMe) on a PCIe Gen4 interface. The combination of EPYC CPU, multiple RTX 3080 GPUs and the NVMe SSD storage made this setup ideal for implementing deep learning models such as the encoder-decoder approach. In a dataset with over two million samples, training the encoder-decoder with a batch size of 256 samples took on average 107 seconds per epoch with this setup. With an average of circa 20 seconds per epoch, the feed-forward network trained faster.

4 Data

CARAVAN is an open-source dataset published by Kratzert et al. (2023). It aims at providing standardized, large-scale and open-access hydrological data. It provides meteorological and streamflow data at a daily frequency as well as static attributes for the included basins. While CARAVAN spans over multiple countries, this paper focuses on US catchments. In the original CAMELS US dataset, as it is maintained by the US National Center for Atmospheric Research (NCAR) (see also Addor et al. 2017 or Newman et al. 2022), there are 671 hydrological basins located in the United States. These basins have land areas between 4 km^2 and 25,000 km^2. Kratzert et al. (2023), however, focused on basins with land area between 100 km^2 and 2,000 km^2, of which there are 482.

Starting with these 482 CAMELS US basins, the data is split randomly into three different subsets. This train-test split is common practice when training machine learning models. Out of 482 basins, about 60 percent or 264 basins are selected to be used during training. The remaining 218 basins are further split into validation and test catchments. Next, the 264 training basins are divided further into training, validation and test periods. For all basins, data begins on January 1st, 1980 and ends on December 30th 2014 yielding a total of 12,783 days per basin. For each of the 264 training basins, the first 60 percent of days, or the days between January 1st 1980 and May 25th 2001 are selected as train periods. From the similar 264 basins, the days between May 26th 2001 and March 12th 2008 or about 20 percent of the days are selected to be the validation set. The last remaining 20 percent, corresponding to days between March 13th 2008 and December 31st 2014 are selected to be in the test dataset. This procedure makes sure that there remain both, days as well as basins which the model has never seen during training. After these splits, the training set contains the first 7,816 days of 264 basins, yielding about two million training samples.

From the 38 available meteorological forcings in CARAVAN, only the following five forcings are passed as inputs to the respective models:

1. Sum of the total precipitation,
2. Mean of the surface net solar radiation mean,
3. Maximum temperature at 2m above ground,
4. Minimum temperature at 2m above ground and
5. Mean of surface pressure.

Fill (2021) added a sine signal as this provides the network with further temporal information and is supposed to support model accuracy. This approach is mimicked here by adding the following, sixth variable:

6. Sine signal with yearly frequency

Finally, the model also receives the streamflow values of the sequence days that are not the target day. This gives a seventh input variable:

7. Streamflow values.

To clarify, an example of a input sequence might be of length 30. If the 31st day is the target day, the model receives 30 days of inputs $x_t \in \mathbb{R}^7$ to predict the streamflow of day 31. For all models, the streamflow from CARAVAN is used as a target variable. In some cases, models were trained to predict changes in streamflow rather than actual levels. For this approach, first differences of the streamflow were calculated.

In addition to these time-varying forcings, also static catchment attributes at the catchment level are used. These are also made available within CARAVAN. Here, all available data from ERA5-Land, which Kratzert et al. (2023) gathered from the Google Earth Engine repository, are used. These are the following ten climate indices:

1. Mean daily precipitation in mm per day,
2. Mean daily potential evaporation in mm per day,
3. Aridity index, ratio of mean potential evaporation and mean precipitation,
4. Fraction of precipitation falling as snow
5. Mean annual moisture index in range $[-1, 1]$,
6. Moisture index seasonality in range $[0, 2]$,
7. Frequency of high precipitation days,
8. Average duration of high precipitation events ,
9. Frequency of low precipitation days and
10. average duration of low precipitation events.

All variables, time-varying and static, were standardized by subtracting their respective mean and dividing by their standard deviation (LeCun et al. 2012). These two summary statistics are calculated in the training set but are used to standardize the validation and test sets as well, a standard practice in the machine learning field. After standardization, the variables are then brought into the applicable shapes to be passed through LSTM networks. Mainly, this means creating sequences of desired length, i.e. splitting the entire 7,816 training days per basin into sequences of length of for example 30 days, 270 days, etc. and then passing these individual sequences through the respective models.

5 Results

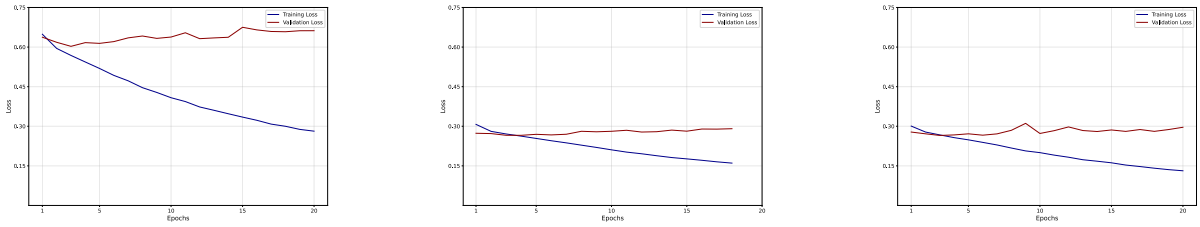
5.1 Encoder-Decoder Part

In the very first approach, the encoder was implemented with one stacked layer, 50 hidden units and using sequence lengths of 30 days. As a target variable, the change in streamflow versus the previous day was chosen. The initial setup already showed an ever decreasing error on the training set. This was read as a positive signal that the model is correctly wired and capable of capturing patterns and/or details in the training data. For the validation data, however, this was not the case. The error was relatively constant across many training epochs. There are multiple possible reasons for this, one being that the model is too complex and, from the very beginning, overfits to the training data. Another possible reason might be that the model architecture is inadequate as these initial hyperparameter settings were taken from Fill (2021) who trained networks for single basins. In contrast, other authors such as Kratzert et al. (2019) who trained models on data from multiple basins used far more complex architectures. Both of these directions were investigated, using less and more complex architectures. While most existing studies stopped training after relatively small number of epochs, models here were trained longer initially to see if the models just take longer to train. Yet the observed behaviour did not change noticeably, even after 500 epochs of training. Across all tested architectures, the training error decreased over the epochs while the validation error remained constant at elevated levels. Even more concerning, the validation error did not change much between training the respective models between 20 and 500 epochs. This observation is illustrated in Figure 5.1 panel (a), which shows the training and validation losses over the first 20 epochs for an encoder-decoder model trained with changes in streamflow as targets. Here, settings from Kratzert et al. (2019) were implemented by choosing sequence lengths of 270, a single stacked layer and 256 hidden units. At the 20th epoch, this model had an MSE of about 0.28 with the training data and an MSE of about 0.66 with the validation data.

None of the existing studies discussed above fitted a model to predict changes in streamflow. Therefore, after the initial experiments, this approach was also changed. To align with the existing literature, the target variable was changed to streamflow levels rather than differences. This decreased both the training as well as the validation errors. However, the observation of a non-decreasing validation error was also observed in case of levels. Existing PyTorch LSTM implementations do not feature variational dropout as discussed in section 2.2.2. Therefore, the model architecture had to be modified using an existing implementation by Viviano (2021) and making minor changes. To make sure nothing in these changes prevented the validation error from falling, also an out-of-the box LSTM was trained to compare its performance. The results are depicted in figure 5.1 panel (b). After 18 epochs, this LSTM had a training MSE of 0.16 and a validation MSE of 0.29.

As this observation of non-decreasing validation errors prevailed also in levels, and with both more complex as well as with simpler architectures, the Kratzert et al. (2019) settings were chosen to proceed with. Their study is closest to work done here. The final implementation, which was used to generate the embeddings, was trained for 20 epochs, as also done by Kratzert et al. (2019). It featured 256 hidden layers and a single stacked layer. The sequence length was also aligned with Kratzert et al. (2019) and again chosen to be 270. At the end of training, this model showed a MSE of 0.13 on the training set and a MSE of just below 0.30 on the validation set. The respective training curves are shown in figure 5.1. In general, the errors had lower values in the case of predicting streamflow levels. Further, the more complex architectures shown in figure 5.1 panels (b) and (c) also had lower levels of errors compared to simpler counterparts.

After generating embeddings for the train and validation sequences with the encoder, they were concatenated with standardized static attributes from the catchments the sequence was taken from. This data was then taken to train feed-forward networks.



(a) Encoder decoder with changes as targets

(b) Out of the box LSTM

(c) Final encoder decoder model used

Figure 5.1 Training and validation losses across selected models.

5.2 Feed-Forward Network Part

In these feed-forward networks, also different levels of complexity were taken into consideration. First, one network with four hidden layers with 256, 128, 64, 32 neurons each, and one with three hidden layers with 128, 64, 32 neurons each were trained. Yet after 200 epochs of training both networks still showed a MSE validation error of about 0.3. This was read as an evidence that the model should again be reduced in terms of complexity. The same was true for a network with 64 and 32 neurons. It achieved a training loss of 0.07 after 200 epochs but the validation loss remained at circa 0.36. Also with 32 and 16 neurons, the validation loss remained at 0.33 after 200 epochs. Even with as low as 8 and 4 neurons per respective hidden layer, the error in the validation set remains constant at roughly 0.32. It still prevails with 4 and 2 neurons where after 200 epochs of training, the training error is down to 0.17 but the error in the validation set remains at 0.34. Table 5.1 shows an overview of the models and their respective performance.

Model	Hidden layers	Layer sizes	Validation error at 200th epoch
I	4	256, 128, 64, 32	0.30
II	3	128, 64, 32	0.30
III	2	64, 32	0.36
IV	2	32, 16	0.33
V	2	8, 4	0.32
VI	2	4, 2	0.34

Table 5.1 Feed forward architectures considered

As all of the models showed more or less a similar level of MSEs, model IV from table 5.1 was chosen as the final model.

5.3 Complete Model

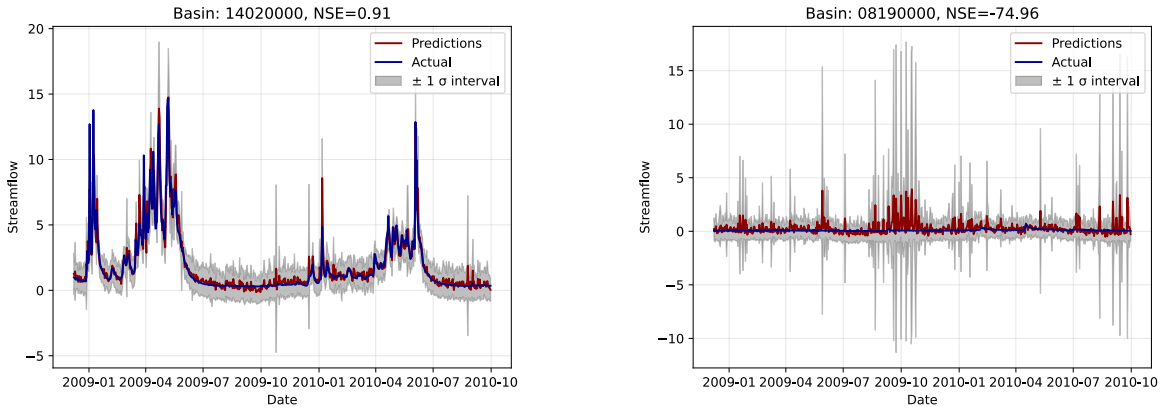
To do inference as described in algorithm 1, one must do multiple passes R through the model. Here, $R = 5$, $R = 10$, and $R = 20$ were considered. Ultimately, $R = 20$ where chosen as early results were most promising with this number of passes. It also relied on Fill (2021) who fitted a MCD model and found one and two iterations to perform significantly worse than five or ten iterations. The dropout rates for the

Statistic		Minimum		Maximum		Mean	Median
		Value	Recorded in basin	Value	Recorded in basin		
NSE		-64626.10	08190500	0.96	13011900	-251.60	0.50
MSE		0.08	04063700	64.39	12141300	4.75	1.82
RMSE		0.27	04063700	8.03	12141300	1.73	1.35
MAD	absolute	0.56	02112360	2.03	09386900	0.80	0.71
	normalized	0.42	05487980	829.63	08190500	5.73	0.60
Variance	absolute	0.69	04127997	27.15	09497980	3.52	2.39
	normalized	0.07	03604000	921,080.55	08190500	3580.67	0.40
Standard Deviation	absolute	0.74	04127997	3.34	09386900	1.15	1.02
	normalized	0.16	03604000	555.21	08190500	3.88	0.45
Eta	absolute	0.79	04122200	10.38	08200000	2.14	1.68
	normalized	0.26	07362100	554.63	08190500	4.33	0.87

Table 5.2 Model performance statistics

encoder LSTM and for the feed forward neural network were chosen to be ten percent. This was in line with the recommendation of Zhu and Laptev (2017) who found that the uncertainty estimation is little changed by varying dropout probabilities.

To get a feeling of the nature of the results, figure 5.2 shows two hydropgraphs, i.e. the streamflow over time. While ground truth streamflow values are depicted in blue, the prediction of the model is shown in red, a $\pm 1\sigma$ interval in grey. Panel (a) shows the hydrograph of a basin in which the model performed well, namely the "Umatilla River above Meacham Creek, NR Gibbon", Oregon, with an NSE of approximately 0.91. Panel (b) shows a negative example of the outlier basin "Nueces Rv at Laguna", Texas, with an NSE of as low as -74.96.



(a) Good model performance

(b) Bad model performance

Figure 5.2 Hydrographs

In the greater picture, the model performed worst in the basin with the id 08190500 in the CARAVAN dataset. This is the "W Nueces Rv nr Brackettville" catchment in Texas. While for this catchment an NSE of -64626.10 was achieved, in the next-best basin, a NSE of -1019.54 was recorded. Excluding the "W Nueces Rv nr Brackettville" catchment, the average NSE already improved to -6.82. Overall, in 32 out of 264 catchments, an NSE below -1.00 was recorded, in 44 an NSE below zero. Table 5.2 shows an overview of these figures. In contrast, the model performed best in catchment 13011900, the "Buffalo Fork AB Lava Creek NR Moran" catchment in Wyoming, where an NSE of about 0.96 was achieved. In a total of 67 catchments, the NSE was above 0.70, in 41 catchments NSEs exceeded the 0.80 threshold. Figure 5.3 shows the cumulative density of the achieved NSEs across basins.

If the predictions are truncated, i.e. setting predictions with values below zero to equal zero, the statistics shown in table 5.2 improve marginally. For example, the average NSE improves to around -241.09.

Without truncation, the model predicted a higher streamflow value than the actually recorded one in about 63.1 percent of test days.

The overall results are broadly in line with prior work done in the field. The basin-specific models trained by Kratzert et al. (2018) achieved a slightly better median NSE of 0.65. The lowest NSE in their model, however, was -0.42 which is far better than the minimum recorded in this work. This, however, is expected as basin-specific models usually achieve better results as they can be fitted to basin-specific circumstances. Yet also the HUC-specific models of Kratzert et al. (2018) achieved a similar level of median NSE compared to basin-specific ones. Data from basins of a similar HUC, however, tend to have a higher correlation among each other, what might be a reason for the good performance of these models. Effectively, this situation gives the model a higher amount of training data. In their most advanced architecture, Kratzert et al. (2018) train models on data from multiple basins and then fine-tune it with basin-specific data only. In this approach, a median NSE of 0.72 is achieved, considerably higher than the median of 0.50 in this work. Fill (2021), who also trained basin-specific models achieved a median NSE of 0.81 in the most advanced setting, also outstripping the encoder-decoder architecture presented here.

In the regional models, Kratzert et al. (2019) also take into account static catchment attributes. Conceptually, this is perhaps the closest study to the work done here. Their model achieved a median NSE of 0.73 and there were six basins with a NSE below zero. This performance is better than the one of the architecture presented here. From this, one can conclude that the model architecture from Zhu and Laptev (2017), as it was implemented and trained here, did not help to improve the performance meaningfully. The biggest difference from the implementation of Kratzert et al. (2019) to the one implemented here is that their best-performing model sees the static attributes at every time-step of the respective sequence. In the architecture presented here, the static attributes are only concatenated with the embedding at the final step, once the input sequence has already been passed through the encoder. This might be the reason for the considerably poorer performance. Other major differences are that Kratzert et al. (2019) had a larger training set. Here, some basins were excluded to see how the model performs on data from unseen basins. Answering this question, unfortunately, was out of scope for now.

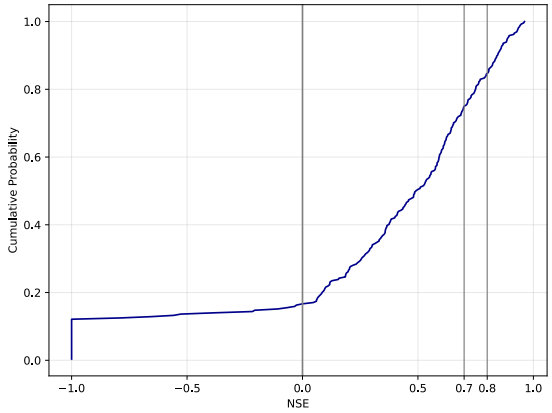


Figure 5.3 Cumulative distribution of achieved NSEs. NSEs below -1.00 are clipped to -1.00 for better scale of the chart.

As the worst performance of the model was achieved in catchments in Texas, this is reminiscent of the Kratzert et al. (2018) hypothesis that a deep learning model performs poorly in basins where there are little day-to-day changes in streamflow. For example, in figure 5.2 panel (b) one can see that the actual streamflow (blue) is more or less constant and close to zero for the entire prediction horizon. The encoder-decoder model, however, sometimes predicts unusually high streamflow which greatly contributes to the poor performance considering the NSE statistic. Kratzert et al. (2018), who trained basin-specific networks, encountered a similar phenomenon. They also reported that performance was worse in more arid basins and especially in basins where the streamflow was rather constant. They note that for such

catchments existing benchmarking models, such as the SAC-SMA+Snow17 in their case, performed even worse than the LSTMs. Their hypothesis for the this behaviour is that flows in these basins are not as highly correlated. Therefore, they assumed that training models involving data of multiple basins helps to mitigate this problem. In fact, Kratzert et al. (2019), who trained single models on multiple catchments, found that three counter-measures helped to achieve better NSE values in those catchments. The first counter measure was including static catchment attributes. While for Kratzert et al. (2019) this had the greatest impact on performance regarding NSE, it appears to be less helpul here. Again, the most obvious difference is that while their model sees the static attribute during every unfolding steps, ours only gets the static attributes in the very final step. The second improvement suggested by Kratzert et al. (2019) was changing the loss-function to be more sensitive towards low streamflow values. This helped but the magnitude of the improvement was far lower than the one achieved from adding static attributes. A third approach was changing the LSTM architecture to include certain input gates for static attributes. Again, the actual improvements were of low magnitude. As all this is centered around arid basins, and the first results from table 5.2 appear in line with this explanation, this was investigated in more detail by inspecting correlations between achieved NSE values and the respective static attributes of a basin. These correlations are captured by scatterplots which are depicted in figure 5.4.

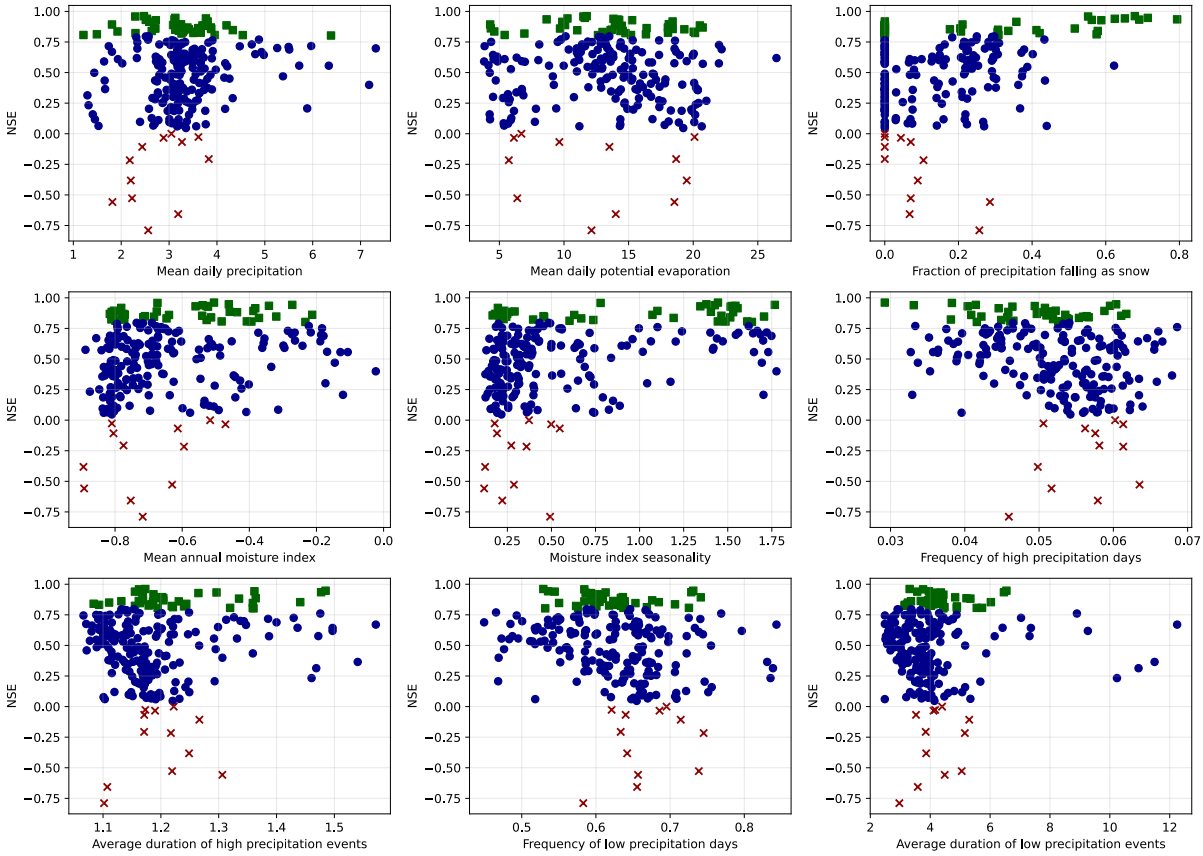


Figure 5.4 Correlations of chatchment NSE values and static catchment attributes. Basins with an NSE below -1.00 have been excluded for better scale. 232 catchments remained. Basins with an NSE of above 0.80 are marked with green squares, basins with NSEs below 0 with red crosses.

Especially the basins with NSEs above 0.80 are distributed over the full ranges of the respective attribute. Basins at the low end of the NSE spectrum are somewhat more arid, for example in terms of mean daily precipitation, and have more high precipitation days. However the picture is far from obvious. Therefore, the limitations in the model of this paper are likely to stem from different sources as the ones put forward in studies by Kratzert et al. (2019) and Kratzert et al. (2018), at least partially.

Regarding the uncertainty, Klotz et al. (2022) suggest some benchmarking metrics to assess the resolution or precision of a model. Those include the mean absolute deviation (MAD), the variance and the standard deviation of predictions. Fill (2021) normalizes these parameters by dividing by the similar statistic calculated on the respective target sequences. Both versions, normalized and absolute are reported in table 5.2. As the encoder-decoder approach from Zhu and Laptev (2017) uses an additional component in the uncertainty quantification, also the final standard deviation from this approach, η, is presented. While the model achieves a median MAD of 0.71 across basins, the median of average η across all basins is 1.68. The MCD model of Klotz et al. (2022) achieved a MAD of 0.39 and a standard deviation of 0.38 of the predictions. The underlying sequences of Klotz et al. (2022), however, had a standard deviation of 2.85. The authors mention that the MCD model was overly confident, especially in high flow periods. Additionally, Klotz et al. (2022) caution that their models take only one source of uncertainty into account. In this regard, the encoder-decoder architecture trained here performed better as the median of normalized η, which is the average η in a basin divided by the standard deviation of the observed streamflows in a basin, was 0.87. The greatest normalized η of 554.63 was observed in basin 08190500, "W Nueces Rv nr Brackettville", Texas, which is also the basin with the worst NSE. The next-smaller normalized η was 65.10, all others were below 28.00. Again, as in the NSE accuracy statistic, the encoder-decoder model seems to suffer from extreme outliers. Naturally, very high values of normalized η are in part also a result of small variance in basin streamflow. This is especially prevalent for basins with streamflow of close to or equal to zero for a long time.

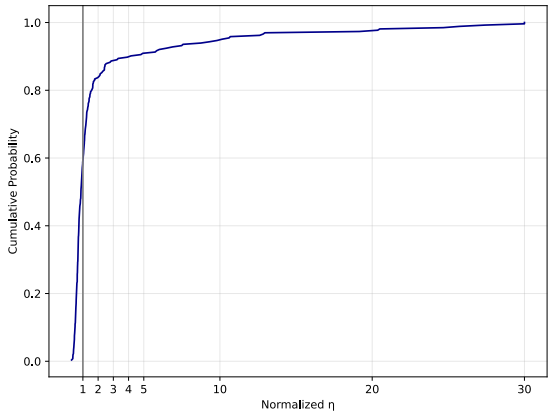


Figure 5.5 Cumulative distribution of normalized η. The two outliers were clipped to a value of 30.00 for better scale.

Figure 5.5 shows a cumulative distribution of η across basins. In his best model configuration, Fill (2021) achieved a normalized MAD of 0.17 and a normalized standard deviation of 0.12. Compared to these two studies, the uncertainty bands produced by the encoder-decoder model are a bit wider. With no actual streamflow value lying outside the $\pm 1\sigma$-interval around the predicted value $\hat{y}_i$, however, the uncertainty bands nicely bracketed the respective predictions.

6 Conclusion

The LSTM-based encoder-decoder approach put forward in Zhu and Laptev (2017) was successfully implemented as a surrogate, data-driven streamflow prediction model in the hydrological realm. In terms of accuracy, the model architecture slightly underperformed existing studies as a median NSE of 0.50 was achieved. The model suffered especially from a handful of outlier catchments where negative NSEs were recorded. In terms of uncertainty quantification, the model performed well. Especially high streamflow events were captured and reliably bracketed by the uncertainty bands. Half of the achieved uncertainty bands were smaller than 0.87 times the standard deviation of the underlying, ground truth streamflow. While there were basins where the model performed excellently in terms of accuracy and resolution, the whole approach seemed to suffer from extreme outliers. In contrast to existing work, bad performing basins were not particularly arid ones when considering static catchment attributed. More likely, the poor performance in the respective basins stemmed from inadequate hyperparameters. A more formal and extensive grid search over possible parameter values should be of great value and might potentially enhance the model performance in these outlier basins. In particular, the phenomenon of non-decreasing validation errors for the encoder should also be studied further. As the overall architecture does not differ so much from existing work, a probable measure for improvement is using a high dropout rate during training. Further, to squeeze every bit of performance out of this architecture, one could also adopt the loss function and make the model more sensitive to low flow regimes. Even though this did not prove to be too useful in other work, it might help this model. Also, training ensembles might further help to alter results. This, however, is unlikely as the model is rather complex and especially small models tend to suffer from initialization problems (Kratzert et al. 2018). To further investigate uncertainty intervals one could research how the use of the dropout rate during performance affects uncertainty intervals in this data and model context. To better be able to compare obtained results, it would also be helpful to train other architectures on the CARAVAN dataset and review their performance on this novel data compilation. From the results presented in this paper, it seems that the strength of the encoder-decoder architecture lies rather on the resolution and uncertainty quantification side. After all, this paper transferred the encoder-decoder architecture to the hydrology realm but there remains ample room for hyperparameter tuning that has the potential to significantly improve model performance.

7 References

Addor, Nans, Guy Nearing, Cristina Prieto, Andrew Newman, Natalyia Le Vine and Martyn Clark, 2018, A ranking of hydrological signatures based on their predictability in space, *Water Resources Research*, vol. 54(11), pp. 8792-8812, https://doi.org/10.1029/2018WR022606

Addor, Nans, Andrew Newman, Naoki Mizukami and Martyn Clark, 2017, The CAMELS data set: catchment attributes and meteorology for large-sample studies, *Hydrology and Earth System Sciences*, vol. 21(10), pp. 5293–5313, https://doi.org/10.5194/hess-21-5293-2017

Andrushia, Diana, N. Anand, Mary Neebha, M. Nazer and Eva Lubloy, 2022, Autonomous detection of concrete damage under fire conditions, *Automation in Construction*, vol. 140(104364), pp. 5293–5313, https://doi.org/10.1016/j.autcon.2022.104364

Bishop, Christopher, 1994, Mixture Density Networks, *Aston Univeristy Neural Computing Research Group*, https://publications.aston.ac.uk/id/eprint/373/1/NCRG_94_004.pdf (last accessed July 7th, 2024)

Blundell, Charles, Julien Cornebise, Koray Kavukcuoglu and Daan Wierstra, 2015, Weight Uncertainty in Neural Networks, *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, https://doi.org/10.48550/arXiv.1505.05424

Duchi, John, Elad Hazan and Yoram Singer, 2011, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, *Journal of Machine Learning Research*, vol. 12, pp. 2121-2159, https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf (last accessed October 16th, 2024)

Esposito, Piero, 2020, BLiTZ - Bayesian Layers in Torch Zoo (a Bayesian Deep Learing library for Torch), *GitHub*, https://github.com/piEsposito/blitz-bayesian-deep-learning/ (last accessed August 7th, 2024)

Fill, Jonas, 2021, Development of the Bayesian Recurrent Neural Network Architectures for Hydrological Time Series Forecasting, *Bachelor Thesis at the Department of Informatics at the Technical Univeristy of Munich*, https://mediatum.ub.tum.de/doc/1631631/1631631.pdf (last accessed September 27th, 2024)

Fortunato, Meire, Charles Blundell and Oriol Vinyals, 2017, Bayesian Recurrent Neural Networks, *12th Women in Machine Learning Workshop (WiML 2017)*, https://doi.org/10.48550/arXiv.1704.02798

Gal, Yarin and Zoubin Ghahramani, 2016, Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning, *International Conference on Machine Learning (ICML)*, https://doi.org/10.48550/arXiv.1506.02142

Gauch, Martin, Frederik Kratzert, Daniel Klotz, Grey Nearing, Jimmy Lin, and Sepp Hochreiter, 2021, Rainfall-runoff prediction at multiple timescales with a single Long Short-Term Memory network, *Hydrology and Earth System Sciences*, vol. 25(4), pp. 2045–2062, https://doi.org/10.5194/hess-25-2045-2021

Gauch, Martin, and Jimmy Lin, 2020, A Data Scientist's Guide to Streamflow Prediction, *arXiv:2006.12975* , https://doi.org/10.48550/arXiv.2006.12975

Gers, Felix, Jürgen Schmidhuber and Fred Cummins, 2000, Learning to Forget: Continual Prediction with LSTM, *Neural Computation*, vol. 12(10), pp. 2451–2471, https://doi.org/10.1162/089976600300015015

Goodfellow, Ian, Yoshua Bengio and Aaron Courville, 2016, Deep Learning, *MIT Press*, http://www.deeplearningbook.org (last accessed October 16th, 2024)

Graves, Alex, Abdelrahman Mohamed and Geoffrey Hinton, 2013, Speech Recognition with Deep Recurrent Neural Networks, *Acoustics, speech and signal processing (ICASSP), 2013, IEEE International Conference*, pp. 6645-6649, https://doi.org/10.48550/arXiv.1303.5778

Harris, Charles, Jarrod Millman, Stefan van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke and Travis Oliphant, 2020, Array programming with NumPy, *Nature*, vol. 585, pp. 1735-1780, https://doi.org/10.1162/neco.1997.9.8.1735

Hochreiter, Sepp and Jürgen Schmidhuber, 1997, Long Short-Term Memory, *Neural Computation*, vol. 9(8), pp. 1735-1780, https://doi.org/10.1162/neco.1997.9.8.1735

Huang, Ruijie, Chenji Wei, Baohua Wang, Jian Yang, Xin Xu, Suwei Wu, and Suqi Huang, 2022, Well performance prediction based on Long Short-Term Memory (LSTM) neural network, *Journal of Petroleum Science and Engineering*, vol. 208(D), https://doi.org/10.1016/j.petrol.2021.109686

Hunter, John, 2007, Matplotlib: A 2D Graphics Environment, *Computing in Science & Engineering*, vol. 9(3), pp. 90-95, https://doi.org/10.1109/MCSE.2007.55

Jayanthi, E. and Vallikannu R, 2022, Enhancing the performance of asymmetric architectures and workload characterization using LSTM learning algorithm, *Advances in Engineering Software*, vol. 173(103266), https://doi.org/10.1016/j.advengsoft.2022.103266

Kingma, Diederik and Jimmy Ba, 2017, Adam: A Method for Stochastic Optimization, *arXiv:1412.6980* , https://doi.org/10.48550/arXiv.1412.6980

Klotz, Daniel, Frederik Kratzert, Martin Gauch, Alden Keefe Sampson, Johannes Brandstetter, Günter Klambauer, Sepp Hochreiter and Grey Nearing, 2022, Uncertainty estimation with deep learning for rainfall–runoff modeling, *Hydrology and Earth System Sciences*, vol. 26(6), pp. 1673–1693, https://doi.org/10.5194/hess-26-1673-2022

Kratzert, Frederik, Grey Nearing, Nans Addor, Tyler Erickson, Martin Gauch, Oren Gilon, Lukas Gudmundsson, Avinatan Hassidim, Daniel Klotz, Sella Nevo, Guy Shalev and Yossi Matias, 2023, Caravan - A global community dataset for large-sample hydrology, *Scientific Data*, vol. 10(61), https://doi.org/10.1038/s41597-023-01975-w

Kratzert, Frederik, Daniel Klotz, Sepp Hochreiter and Grey Nearing, 2021, A note on leveraging synergy in multiple meteorological data sets with deep learning for rainfall–runoff modeling, *Hydrology and Earth System Sciences*, vol. 25(5), pp. 2685–2703, https://doi.org/10.5194/hess-25-2685-2021

Kratzert, Frederik, Daniel Klotz, Guy Shalev, Günter Klambauer, Sepp Hochreiter and Grey Nearing, 2019, Towards learning universal, regional, and local hydrological behaviors via machine learning applied to large-sample datasets, *Hydrology and Earth System Sciences*, vol. 23(12), pp. 5089–5110, https://doi.org/10.5194/hess-23-5089-2019

Kratzert, Frederik, Daniel Klotz, Claire Brenner, Karsten Schulz, and Mathew Herrnegger, 2018, Rainfall–runoff modelling using Long Short-Term Memory (LSTM) networks, *Hydrology and Earth System Sciences*, vol. 22(11), pp. 6005-6022, https://doi.org/10.5194/hess-22-6005-2018

Kumar, Rohini, Luis Samaniego and Sabine Attinger, 2013, Implications of distributed hydrologic model parameterization on water fluxes at multiple scales and locations, *Water Resources Research*, vol. 49(1), pp. 360–379, https://doi.org/10.1029/2012WR012195

LeCun, Yann, Leon Bottou, Genevieve Orr and Klaus-Robert Müller, 2012, Efficient Backprop, *Neural Networks: Tricks of the Trade*, Springer, pp. 9-48, https://link.springer.com/chapter/10.1007/978-3-642-35289-8_3 (last accessed August 9th 2024)

Liang, Liu, Guangguang Xu, Yun Wang, Limei Wang and Jian Liu, 2024, Battery temperature estimation at wide C-rates using the LSTM model based on polarization characteristics, *Journal of Energy Storage*, vol. 101(B), https://doi.org/10.1016/j.est.2024.113941

Liang, Xu, Dennis Lettenmaier, Eric Wood, and Stephen Burges, 1994, A simple hydrologically based model of land surface water and energy fluxes for general circulation models, *Journal of Geophysical Research Atmospheres*, vol. 99(D7), pp. 14415-14428, https://doi.org/10.1029/94JD00483

McKinney, Wes, 2010, Data structures for statistical computing in python, *Proceedings of the 9th Python in Science Conference*, vol. 445, pp. 56-61, https://doi.org/10.25080/Majora-92bf1922-00a

Morris, Max, 1991, Factorial Sampling Plans for Preliminary Computational Experiments, *Technometrics*, vol. 33(2), pp. 161-174, https://doi.org/10.2307/1269043

Nash, James, and John Sutcliffe, 1970, River flow forecasting through conceptual models part I - A discussion of principles, *Journal of Hydrology*, vol. 10(3), pp. 282-290, https://doi.org/10.1016/0022-1694(70)90255-6

Nevo, Sella, Efrat Morin, Adi Gerzi Rosenthal, Asher Metzger, Chen Barshai, Dana Weitzner, Dafi Voloshin, Frederik Kratzert, Gal Elidan, Gideon Dror, Gregory Begelman, Grey Nearing, Guy Shalev, Hila Noga, Ira Shavitt, Liora Yuklea, Moriah Royz, Niv Giladi, Nofar Peled Levi, Ofir Reich, Oren Gilon, Ronnie Maor, Shahar Timnat, Tal Shechter, Vladimir Anisimov, Yotam Gigi, Yuval Levin, Zach Moshe, Zvika Ben-Haim, Avinatan Hassidim and Yossi Matias, 2022, Flood forecasting with machine learning models in an operational framework, *Hydrology and Earth System Sciences*, vol. 26(15), pp. 4013–4032, https://doi.org/10.5194/hess-26-4013-2022

Newman, Andrew, Kevin Sampson, Martyn Clark, A. Bock, J. Viger, D. Blodgett, Nans Addor and M. Mizukami, 2022, CAMELS: Catchment Attributes and MEteorology for Large-sample Studies, *UCAR/NCAR - GDEX*, https://gdex.ucar.edu/dataset/camels.html (last accessed July 7th, 2024)

Newman, Andrew, M. P. Clark, K. Sampson, A. Wood, L. E. Hay, A. Bock, R. J. Viger, D. Blodgett, L. Brekke, J. R. Arnold, T. Hopson, and Q. Duan, 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*, vol. 19(1), pp. 209–223, https://doi.org/10.5194/hess-19-209-2015

Panja, Palash, Wei Jia and Brian McPherson, 2022, Prediction of well performance in SACROC field using stacked Long Short-Term Memory (LSTM) network, *Expert Systems with Applications*, vol. 205(117670), https://doi.org/10.1016/j.eswa.2022.117670

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai and Soumith Chintala, 2019, PyTorch: An Imperative Style, High-Performance Deep Learning Library, *arXiv:1912.01703* , https://doi.org/10.48550/arXiv.1912.01703

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay, 2011, Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, vol. 12(85), pp. 2825-2830, https://www.jmlr.org/papers/v12/pedregosa11a.html (last accessed August 1st, 2024)

Rumelhart, David, Geoffrey Hinton and Ronald Williams, 1986, Learning representations by back-propagating errors, *Nature*, vol. 323, pp. 533-536, https://doi.org/10.1038/323533a0

Samaniego, Luis, Rohini Kumar, and Sabine Attinger, 2010, Multiscale parameter regionalization of a grid-based hydrologic model at the mesoscale, *Water Resources Research*, vol. 46(5), W05523, https://doi.org/10.1029/2008WR007327

Shen, Chaopeng, 2018, A Transdisciplinary Review of Deep Learning Research and Its Relevance for Water Resources Scien-

tists, *Water Resources Research*, vol. 54(11), pp. 8558-8593, https://doi.org/10.1029/2018WR022643

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov, 2014, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *Journal of Machine Learning Research*, vol. 15(56), pp. 1929-1958, http://jmlr.org/papers/v15/srivastava14a.html (last accessed September 26th, 2024)

Sutskever, Ilya, Oriol Vinyals and Quoc Le, 2014, Sequence to Sequence Learning with Neural Networks, *arXiv:1409.3215* , https://doi.org/10.48550/arXiv.1409.3215

van Rossum, Guido, 1995, Python reference manual, *Centrum Wiskunde & Informatica (CWI), Department of Computer Science, Tech Report*, https://ir.cwi.nl/pub/5008/05008D.pdf (last accessed October 3rd, 2024)

Viviano, Jospeh, 2021, Variational Dropout for LSTMs, *GitHub*, https://github.com/josephdviviano/lstm-variational-dropout/tree/master (last accessed October 3rd, 2024)

Wang, Yongjie, Changhong Zhan, Guanghao Li, Dongjie Zhang and Xueying Han, 2023, Physics-guided LSTM model for heat load prediction of buildings, *Energy and Buildings*, vol. 294(113169), https://doi.org/10.1016/j.enbuild.2023.113169

Xu, Boyan, Ching Pooi, Kar Tan, Shujuan Huang, Xueqing Shi and How Ng, 2023, A novel long short-term memory artificial neural network (LSTM)-based soft-sensor to monitor and forecast wastewater treatment performance, *Journal of Water Process Engineering*, vol. 54(104041), https://doi.org/10.1016/j.jwpe.2023.104041

Zhu, Lingxue and Nikolay Laptev, 2017, Deep and Confident Prediction for Time Series at Uber, *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, https://doi.org/10.1109/ICDMW.2017.19