MC-LSTM: Appendix

A. Notation Overview

Most of the notation used throughout the paper, is summarized in Tab. A.1.

B. Experimental Details

In the following, we provide further details on the experimental setups.

B.1. Neural Arithmetic

Neural networks that learn arithmetic operations have recently come into focus (Trask et al., 2018; Madsen & Johansen, 2020). Specialized neural modules for arithmetic operations could play a role for complex AI systems since cognitive studies indicate that there is a part of the brain that enables animals and humans to perform basic arithmetic operations (Nieder, 2016; Gallistel, 2018). Although this primitive number processor can only perform approximate arithmetic, it is a fundamental part of our ability to understand and interpret numbers (Dehaene, 2011).

B.1.1. Details on Datasets

We consider the *addition problem* that was proposed in the original Long Short-Term Memory (LSTM) paper (Hochreiter & Schmidhuber, 1997). We chose input values in the range $[0, 0.5]$ in order to be able to use the fast standard implementations of LSTM. For this task, 20 000 samples were generated using a fixed random seed to create a dataset, which was split in 50% training and 50% validation samples. For the test data, 1 000 samples were generated with a different random seed.

A definition of the *static arithmetic* task is provided by (Madsen & Johansen, 2020). The following presents this definition and its extension to the *recurrent arithmetic* task (c.f. Trask et al., 2018).

The input for the static version is a vector, $\boldsymbol{x} \in \mathcal{U}(1,2)^{100}$, consisting of numbers that are drawn randomly from a uniform distribution. The target, y, is computed as

$$y = \left(\sum_{k=a}^{a+c} x_k \right) \square \left(\sum_{k=b}^{b+c} x_k \right),$$

where $c \in \mathbb{N}$, $a \leq b \leq a + c \in \mathbb{N}$ and $\square \in \{+, -, \cdot\}$. For the recurrent variant , the input consists of a sequence of T vectors, denoted by $\boldsymbol{x}^t \in \mathcal{U}(1,2)^{10}, t \in \{1, \dots, T\}$, and the labels are computed as

$$y = \left(\sum_{t=1}^{T} \sum_{k=a}^{a+c} x_k^t \right) \square \left(\sum_{t=1}^{T} \sum_{k=b}^{b+c} x_k^t \right).$$

For these experiments, no fixed datasets were used. Instead, samples were generated on the fly. For the recurrent tasks, 2 000 000 batches of 128 problems were created and for the static tasks 500 000 batches of 128 samples were used in the addition and subtraction tasks and 3 000 000 batches for multiplication. Note that since the subsets overlap, i.e., inputs are re-used, this data does not have mass conservation properties.

For a more detailed description of the *MNIST addition* data, we refer to (Trask et al., 2018) and the appendix of (Madsen & Johansen, 2020).

B.1.2. Details on Hyperparameters.

For the *addition problem*, every network had a single hidden layer with 10 units. The output layer was a linear, fully connected layer for all Mass-Conserving LSTM (MC-LSTM) and LSTM variants. The Neural Addition Unit (NAU) (Madsen & Johansen, 2020) and Neural ALU (NALU)/Neural Accumulator (NAC) (Trask et al., 2018) networks used their corresponding output layer. Also, we used a more common L_2 regularization scheme with low regularization constant (10^{-4}) to keep the weights ternary for the NAU, rather than the strategy used in the reference implementation from Madsen & Johansen (2020). Optimization was done using Adam (Kingma & Ba, 2015) for all models. The initial learning rate was selected from $\{0.1, 0.05, 0.01, 0.005, 0.001\}$ on the validation data for each method individually. All methods were trained for 100 epochs.

The weight matrices of LSTM were initialized in a standard way, using orthogonal and identity matrices for the forward and recurrent weights, respectively. Biases were initialized to be zero, except for the bias in the forget gate, which was initialized to 3. This should benefit the gradient flow for the first updates. Similarly, MC-LSTM is initialized so that the redistribution matrix (cf. Eq. 7) is (close to) the identity matrix. Otherwise we used orthogonal initialization (Saxe et al., 2014). The bias for the output gate was initialized to -3. This stimulates the output gates to stay closed (keep mass in the system), which has a similar effect as setting the forget gate bias in LSTM. This practically holds for all subsequently described experiments.

Table A.1. Symbols and notations used in this paper.

Definition	Symbol/Notation	Dimension
mass input at timestep t	$\boldsymbol{x}^t$ or x^t	M or 1
auxiliary input at timestep t	$\boldsymbol{a}^{\mathrm{t}}$	L
cell state at timestep t	$\boldsymbol{c}^t$	K
limit of sequence of cell states	$\boldsymbol{c}^\infty$	
hidden state at timestep t	$\boldsymbol{h}^t$	K
redistribution matrix	$\boldsymbol{R}$	$K \times K$
input gate	$\boldsymbol{i}$	K
output gate	$\boldsymbol{o}$	K
mass	$\boldsymbol{m}$	K
input gate weight matrix	$\boldsymbol{W}_\mathrm{i}$	$K \times L$
input gate weight matrix	$\boldsymbol{W}_\mathrm{o}$	$K \times L$
output gate weight matrix	$\boldsymbol{U}_\mathrm{i}$	$K \times K$
output gate weight matrix	$\boldsymbol{U}_\mathrm{o}$	$K \times K$
identity matrix	$\boldsymbol{K}$	$K \times K$
input gate bias	$\boldsymbol{b}_\mathrm{i}$	K
output gate bias	$\boldsymbol{b}_\mathrm{o}$	K
arbitrary differentiable function	f	
hypernetwork function (conditioner)	g	
redistribution gate bias	$\boldsymbol{B}_\mathrm{R}$	$K \times K$
stored mass	m_c	
mass efflux	m_h	
limit of series of mass inputs	m_x^∞	
timestep index	t	
an arbitrary timestep	τ	
last timestep of a sequence	T	
redistribution gate weight tensor	$\boldsymbol{\mathsf{W}}_\mathrm{r}$	$K \times K \times L$
redistribution gate weight tensor	$\boldsymbol{\mathsf{U}}_\mathrm{r}$	$K \times K \times K$
arbitrary feature index	a	
arbitrary feature index	b	
arbitrary feature index	c	

For the *recurrent arithmetic tasks*, we tried to stay as close as possible to the setup that was used by Madsen & Johansen (2020). This means that all networks had again a single hidden layer. The NAU, Neural Multiplication Unit (NMU) and NALU networks all had two hidden units and, respectively, NAU, NMU and NALU output layers. The first, recurrent layer for the first two networks was a NAU and the NALU network used a recurrent NALU layer. For the exact initialization of NAU and NALU, we refer to (Madsen & Johansen, 2020).

The MC-LSTM models used a fully connected linear layer with L_2-regularization for projecting the hidden state to the output prediction for the addition and subtraction tasks. A free linear layer was used to compensate for the fact that the data does not have mass-conserving properties. However, it is important to note that the mass conservation in MC-LSTM is still necessary to solve this task. For the multiplication problem, we used a multiplicative, non-recurrent variant of MC-LSTM with an extra scalar parameter to allow the conserved mass to be re-scaled if necessary. This multiplicative layer is described in more detail in Appendix B.1.3.

Whereas the addition could be solved with two hidden units, MC-LSTM needed three hidden units to solve both subtraction and multiplication. This extra unit, which we refer to as the *trash cell*, allows MC-LSTMs to get rid of excessive mass that should not influence the prediction. Note that, since the mass inputs are vectors, the input gate has to be computed in a similar fashion as the redistribution matrix. Adam was again used for the optimization. We used the same learning rate (0.001) as Madsen & Johansen (2020) to train the NAU, NMU and NALU networks. For MC-LSTM the learning rate was increased to 0.01 for addition and subtraction and 0.05 for multiplication after a manual search on the validation set. All models were trained for two million update steps.

In a similar fashion, we used the same models from Madsen & Johansen (2020) for the *MNIST addition* task. For MC-LSTM, we replaced the recurrent NAU layer with a MC-LSTM layer and the output layer was replaced with a fully connected linear layer. In this scenario, increasing the learning rate was not necessary. This can probably be explained by the fact that training Convolutional Neural Network (CNN) to regress the MNIST images is the main challenge during learning. We also used a standard L_2-regularization on the outputs of CNN instead of the implementation proposed in (Madsen & Johansen, 2020) for this task.

B.1.3. Static Arithmetic

This experiment should enable a more direct comparison to the results from Madsen & Johansen (2020) than the recurrent variant. The data for the static task is equivalent to that of the recurrent task with sequence length one. For more details on the data, we refer to Appendix B.1.1 or (Madsen & Johansen, 2020).

Since the static task does not require a recurrent model, we discarded the redistribution matrix in MC-LSTM. The result is a layer with only input and output gates, which we refer to as a Mass-Conserving Fully Connected (MC-FC) layer. We compared this model to the results reported in (Madsen & Johansen, 2020), using the code base that accompanied the paper. All NALU and NAU networks had a single hidden layer. Similar to the recurrent task, MC-LSTM required two hidden units for addition and three for subtraction. Mathematically, an MC-FC with K hidden neurons and M inputs can be defined as MC-FC : $\mathbb{R}^M \to \mathbb{R}^K : \boldsymbol{x} \mapsto \boldsymbol{y}$, where

$$\boldsymbol{y} = \operatorname{diag}(\boldsymbol{o}) \cdot \boldsymbol{I} \cdot \boldsymbol{x} \quad \boldsymbol{I} = \operatorname{softmax}(\boldsymbol{B}_I) \quad \boldsymbol{o} = \sigma(\boldsymbol{b}_o),$$

where the softmax operates on the row dimension to get a column-normalized matrix, $\boldsymbol{I}$, for the input gate.

Using the $\log$-$\exp$ transform (c.f. Trask et al., 2018), a multiplicative MC-FC with scaling parameter, $\boldsymbol{\alpha}$, can be constructed as follows: $\exp(\text{MC-FC}(\log(\boldsymbol{x})) + \boldsymbol{\alpha})$. The scaling parameter is necessary to break the mass conservation when it is not needed. By replacing the output layer with this multiplicative MC-FC, it can also be used to solve the multiplication problem. This network also required three hidden neurons. This model was compared to a NMU network with two hidden neurons and NALU network.

All models were trained for two million updates with the Adam optimizer (Kingma & Ba, 2015). The learning rate was set to 0.001 for all networks, except for the MC-FC network, which needed a lower learning rate of 0.0001, and the multiplicative MC-FC variant, which was trained with learning rate 0.01. These hyperparameters were found using a manual search.

Since the input consists of a vector, the input gate predicts a left-stochastic matrix, similar to the redistribution matrix. This allows us to verify generalization abilities of the inductive bias in MC-LSTMs. The performance was measured in a similar way as for the recurrent task, except that generalization was tested over the range of the input values (Madsen & Johansen, 2020). Concretely, the models were trained on input values in $[1, 2]$ and tested on input values in the range $[2, 6]$. Table B.2 shows that MC-FC is able to match or outperform both NALU and NAU on this task.

B.1.4. Comparison with Time-dependent MC-LSTM

We used MC-LSTM with a time-independent redistribution matrix, as in Eq. (7), to solve the addition problem. This resembles another form of inductive bias, since we know that no redistribution across cells is necessary to solve this

Table B.2. Results for the static arithmetic task. MC-FC is a mass-conserving variant of MC-LSTM based on fully-connected layers for non-recurrent tasks. MC-FCs for addition and subtraction/multiplication have two and three neurons, respectively. Error bars represent 95% confidence intervals.

	addition		subtraction		multiplication	
	success rate[a]	updates[b]	success rate[a]	updates[b]	success rate[a]	updates[b]
MC-FC	$\mathbf{100\%}\,^{+0\%}_{-4\%}$	$2.1 \cdot 10^5$	$\mathbf{100\%}\,^{+0\%}_{-4\%}$	$1.6 \cdot 10^5$	$\mathbf{100\%}\,^{+0\%}_{-4\%}$	$1.4 \cdot 10^6$
NAU / NMU	$\mathbf{100\%}\,^{+0\%}_{-4\%}$	$1.8 \cdot 10^4$	$\mathbf{100\%}\,^{+0\%}_{-4\%}$	$5.0 \cdot 10^3$	$98\%\,^{+1\%}_{-5\%}$	$1.4 \cdot 10^6$
NAC	$\mathbf{100\%}\,^{+0\%}_{-4\%}$	$2.5 \cdot 10^5$	$\mathbf{100\%}\,^{+0\%}_{-4\%}$	$9.0 \cdot 10^3$	$31\%\,^{+10\%}_{-8\%}$	$2.8 \cdot 10^6$
NALU	$14\%\,^{+8\%}_{-5\%}$	$1.5 \cdot 10^6$	$14\%\,^{+8\%}_{-5\%}$	$1.9 \cdot 10^6$	$0\%\,^{+4\%}_{-0\%}$	–

[a] Percentage of runs that generalized to a different input range.

[b] Median number of updates necessary to solve the task.

problem and it results also in a more efficient model, because less parameters have to be learned. However, for the sake of flexibility, we also verified that it is possible to use the more general time-dependent redistribution matrix (cf. Eq. 8). The results of this experiment can be found in Table B.3.

Although the performance of MC-LSTM with time-dependent redistribution matrix is slightly worse than that of the more efficient MC-LSTM variant, it still outperforms all other models on the generalisation tasks. This can partly be explained by the fact that is harder to train a time-dependent redistribution matrix, while the training budget is limited to 100 epochs.

B.1.5. Comparison with Normalized and Unitary Networks

In order to account for the limited range of LSTMs, normalization techniques can be used to keep the data within a manageable range. Therefore, we also compared MC-LSTM to an LSTM with layer normalization (Ba et al., 2016). Although the layer normalization improves the generalization performance, it does not match the performance of MC-LSTM (see Table B.3).

Whereas MC-LSTM preserves the L_1 norm, unitary RNNs (Arjovsky et al., 2016) preserve the L_2 norm. To make sure that our inductive bias on the L_1 norm is justified, we directly compared MC-LSTM to a unitary RNN. We adopted the hyperparameters from Arjovsky et al. (2016) and tried fine-tuning them, but were unable to reproduce the results on the addition problem. Nevertheless, we include the generalization performance of the best performing unitary RNN in table B.3.

B.1.6. Qualitative Analysis of the MC-LSTM Models Trained on Arithmetic Tasks

Addition Problem. To reiterate, we used MC-LSTM with 10 hidden units and replaced the linear output layer by a simple summation. The model has to learn to sum all mass inputs of the timesteps, where the auxiliary input (the *marker*) equals $a^t = 1$, and ignore all other values. At the final timestep — where the auxiliary input equals $a^t = -1$ — the network should output the sum of all previously marked mass inputs.

In our experiment, the model has learned to store the marked input values in a single cell, while all other mass inputs mainly end up in a single, different cell. That is, a single cell learns to accumulate the inputs to compute the solution and the other cells are used as *trash cells*. In Fig. B.1, we visualize the cell states for a single input sample over time, where the orange and the blue line denote the mass accumulator and the main trash cell, respectively.

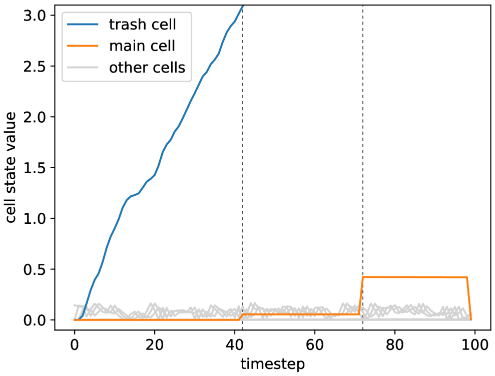

Figure B.1. MC-LSTM cell states over time for model trained to solve the *addition problem* (see Appendix B.1.1). Each line denotes the value of one particular cell over time, while the two vertical grey indicator lines denote the timesteps, where the auxiliary input was 1 (i.e., which numbers in the sequence have to be added).

We can see that at the last time step — where the network is queried to return the accumulated sum — the value of

Table B.3. Performance of different models on the LSTM addition task in terms of the MSE. MC-LSTM significantly (all p-values below .05) outperforms its competitors, LSTM (with high initial forget gate bias), NALU, NAU a layer-normalized LSTM (LN-LSTM) and unitary RNNs (URNN). Error bars represent 95%-confidence intervals across 100 runs.

	reference[a]	seq length[b]	input range[c]	count[d]	combo[e]	`NaN`[f]
MC-LSTM†	0.013 ± 0.004	0.022 ± 0.010	2.6 ± 0.8	2.2 ± 0.7	13.6 ± 4.0	0
MC-LSTM	**0.004** ± 0.003	**0.009** ± 0.004	**0.8** ± 0.5	**0.6** ± 0.4	**4.0** ± 2.5	0
LSTM	0.008 ± 0.003	0.727 ± 0.169	21.4 ± 0.6	9.5 ± 0.6	54.6 ± 1.0	0
LN-LSTM	0.026 ± 0.003	0.055 ± 0.010	24.5 ± 0.3	7.5 ± 0.2	62.0 ± 0.5	0
URNN	0.043 ± 0.001	0.139 ± 0.133	99.9 ± 63.8	7.0 ± 0.1	88.1 ± 3.4	0
NALU	0.060 ± 0.008	0.059 ± 0.009	25.3 ± 0.2	7.4 ± 0.1	63.7 ± 0.6	93
NAU	0.248 ± 0.019	0.252 ± 0.020	28.3 ± 0.5	9.1 ± 0.2	68.5 ± 0.8	24

[a] training regime: summing 2 out of 100 numbers between 0 and 0.5.
[b] longer sequence lengths: summing 2 out of 1 000 numbers between 0 and 0.5.
[c] more *mass* in the input: summing 2 out of 100 numbers between 0 and 5.0.
[d] higher number of summands: summing 20 out of 100 numbers between 0 and 0.5.
[e] combination of previous scenarios: summing 10 out of 500 numbers between 0 and 2.5.
[f] Number of runs that did not converge.
† MC-LSTM with time-dependent redistribution matrix.

this mass accumulator drops to zero, i.e., the output gate is completely open. Note that this would not be the case for a model with a fully connected layer. After all, the fully connected layer can arbitrarily scale the output of the MC-LSTM layer, which allows the output gate to open only partially. Apart from this distinction in the last timestep, the cell states for both models behave the same way. For all other cells (grey lines), the output gate at the last time step is zero. This illustrates nicely how the model output is only determined by the value of the single cell that acted as accumulator of the marked values (orange line).

Also note the accumulating behaviour of the main trash cell (blue line). This can become a problem for very long sequences, because the logistic sigmoid in the output gate can never be perfectly 1 or 0. This means that if the value in the trash cell grows too large, it might effectively leak into the output. However, this undesired behaviour could be countered by, e.g., L_2 regularisation on the cell states, or in case of continuous prediction, adding an extra output as outlet for unnecessary mass (see Sec. B.4.2). This should push the network to dump the trash cell to the output at timesteps where the output is not used.

Recurrent Arithmetic. In the following we take a closer look at the solution that is learned with MC-LSTM. Concretely, we look at the weights of a MC-LSTM model that successfully solves the following recurrent arithmetic task:

$$y = \sum_{t=1}^{T}(x_6^t + x_7^t) \square \sum_{t=1}^{T}(x_7^t + x_8^t),$$

where $\square \in \{-, +\}$, given a sequence of input vectors $\boldsymbol{x}^t \in \mathbb{R}^{10}$ (the only purpose of the colors is to provide an aid to readers). We highlight the following observations:

1. For the addition task (i.e., $\square \equiv +$), MC-LSTM has two units (see Appendix B.1.2 for details on the experiments). Trask et al. (2018); Madsen & Johansen (2020) fixed the number of hidden units to two with the idea that each unit can learn one term of the addition operation ($\square$). However, if we take a look at the input gate of our model, we find that the first cell is used to accumulate $(x_1^t + \ldots + x_5^t + 0.5x_6^t + 0.5x_8^t + x_9^t + x_{10}^t)$ and the second cell collects $(0.5x_6^t + x_7^t + 0.5x_8^t)$. Since the learned redistribution matrix is the identity matrix, these accumulators operate individually.

   This means that, instead of computing the individual terms, MC-LSTM directly computes the solution, scaled by a factor ½ in its second cell. The first cell accumulates the rest of the mass, which it does not need for the prediction. In other words, it operates as some sort of *trash cell*. Note that due to the mass-conservation property, it would be impossible to compute each side of the operation individually. After all, x_7^t appears on both sides of the central operation ($\square$), and therefore the data is not mass conserving.

   The output gate is always open for the *trash cell* and closed for the other cell, indicating that redundant mass is discarded through the output of the MC-LSTM in every timestep and the scaled solution is properly accumulated. However, in the final timestep — when the prediction is to be made, the output gate for the trash cell is closed and opened for the other cell. That is, the accumulated solution is passed to the final linear layer, which scales the output of MC-LSTM by a factor of two to get the correct solution.

2. For the subtraction task (i.e., $\square \equiv -$), a similar behavior can be observed. In this case, the final model requires three units to properly generalize. The first two cells accumulate x_6^t and x_8^t, respectively. The last cell operates as *trash cell* and collects $(x_1^t + \ldots + x_5^t + x_7^t + x_9^t + x_{10}^t)$. The redistribution matrix is the identity matrix for the first two cells. For the *trash cell*, equal parts (0.4938) are redistributed to the two other cells. The output gate operates in a similar fashion as for addition. Finally, the linear layer computes the difference between the first two cells with weights 1, -1 and the *trash cell* is ignored with weight 0.

   Although MC-LSTM with two units was not able to generalize well enough for the Madsen & Johansen (2020) benchmarks, it did turn out to be able to provide a reasonable solution (albeit with numerical flaws). With two cells, the network learned to store $(0.5x_1^t + \ldots + 0.5x_5^t + x_6^t + 0.5x_7^t + 0.5x_9^t + 0.5x_{10}^t)$ in one cell, and $(0.5x_1^t + \ldots + 0.5x_5^t + 0.5x_7^t + x_8^t + 0.5x_9^t + 0.5x_{10}^t)$ in the other cell. With a similar linear layer as for the three-unit variant, this solution should also compute a correct solution for the subtraction task.

B.2. Inbound-outbound Traffic Forecast

Traffic forecasting considers a large number of different settings and tasks (Tedjopurnomo et al., 2020). For example whether the physical network topology of streets can be exploited by using graph neural networks combined with LSTMs (Cui et al., 2019). Within traffic forecasting mass conservation translates to a *conservation-of-vehicles* principle. Generally, models that adhere to this principle are desired (Vanajakshi & Rilett, 2004; Zhao et al., 2017) since they could be useful for long-term forecasts. Many recent benchmarking datasets for traffic forecasts are usually uni-directional and are measured at few streets. Thus conservation laws cannot be directly applied (Tedjopurnomo et al., 2020).

We demonstrate how MC-LSTM can be used in traffic forecasting settings. A typical setting for vehicle conservation is when traffic counts for inbound and outbound roads of a city are available. In this case, all vehicles that come from an inbound road must either be within a city or leave the city on an outbound road. The setting is similar to passenger flows in inbound and outbound metro (Liu et al., 2019), where LSTMs have also prevailed. We were able to extract such data from a recent dataset based on GPS-locations (Kreil et al., 2020) of vehicles at a fine geographic grid around cities, which represents good approximation of a vehicle conserving scenario.

An approximately mass-conserving traffic dataset Based on the data for the traffic4cast 2020 challenge (Kreil et al., 2020), we constructed a dataset to model inbound and outbound traffic of three different cities: Berlin, Istanbul and Moscow. The original data consists of 181 sequences of multi-channel images encoding traffic volume and speed for every five minutes in four (binned) directions. Every sequence corresponds to a single day in the first half of the year. In order to get the traffic flow from the multi-channel images at every timestep, we defined a frame around the city and collected the traffic-volume data for every pixel on the border of this frame. This is illustrated in Fig. 3. For simplicity, we ignored the fact that a single-pixel frame might have issues with fast-moving vehicles.

By taking into account the direction of the vehicles, the inbound and outbound traffic can be combined for every pixel on the border of our frame. To get a more tractable dataset, we additionally combined the pixels of the four edges of the frame to end up with eight values: four values for the incoming traffic, i.e: one for each border of the frame, and four values for the outgoing traffic. The inbound traffic would be the *mass* input for MC-LSTM and the target outputs are the outbound traffic along the different borders. The auxiliary input is the current daytime, encoded as a value between zero and one.

To model the sparsity that is often available in other traffic counting problems, we chose three time-slots (6 am, 12 pm and 6 pm) for which we use fifteen minutes of the actual measurements — i.e., three timesteps. This could for example simulate the deployment of mobile traffic counting stations. The other inputs are imputed by the average inbound traffic over the training data, which consists of 181 days. Outputs are only available when the actual measurements are used. This gives a total of 9 timesteps per day on which the loss can be computed. For training, this dataset is randomly split in 85% training and 15% validation samples.

During inference, all 288 timesteps of the inbound and outbound measurements are used to find out which model learned the traffic dynamics from the sparse training data best. For this purpose, we used the 18 sequences of validation data from the original dataset as test set, which are distributed across the second half of the year. In order to enable a fair comparison between LSTM and MC-LSTM, the data for LSTM was normalized to zero mean and unit variance for training and inference (using statistics from the training data). MC-LSTM does not need this pre-processing step and is fed the raw data.

Model and Hyperparameters For the traffic prediction, we used LSTM followed by a fully connected layer as base-line (c.f. Zhao et al., 2017; Liu et al., 2019). For MC-LSTM, we chose to enforce end-to-end mass conservation by using a MC-FC output layer, which is described in detail in Appendix B.1.3. For the initialization of the models, we refer to the details of the arithmetic experiments in Appendix B.1.

For each model and for each city, the best hyperparameters were found by performing a grid search on the validation data. This means that the hyperparameters were chosen to minimize the error on the nine 5-minute intervals. For all models, the number of hidden neurons was chosen from $\{10, 50, 100\}$ and for the learning rate, the options were $\{0.100, 0.050, 0.010, 0.005, 0.001\}$. All models were trained for 2 000 epochs using the Adam optimizer (Kingma & Ba, 2015). Additionally, we considered values in $\{0, 5\}$ for the initial value of the forget gate bias in LSTM. For MC-LSTM, the extra hyperparameters were the initial cell state value ($\in \{0, 100\}$) — i.e., how much cars are in each memory cell at timestep zero — and whether or not the initial cell state should be trained via backpropagation. The results of the hyperparameter search can be found in Tab. B.4.

The idea behind tuning the initial cell state, is that unlike with LSTM, the cell state in MC-LSTM directly reflects the number of cars that can drive out of a city during the first timesteps. If the initial cell state is too high or too low, this might negatively affect the prediction capabilities of the model. If it would be possible to estimate the number of cars in a city at the start of the sequence, this could also be used to get better estimates for the initial cell state. However, from the results of the hyperparameter search (see Tab. B.4), we might have overestimated the importance of these hyperparameters.

Results. All models were evaluated on the test data, using the checkpoint after 2 000 epochs for fifty runs. An example of what the predictions of both models look like for an arbitrary day in an arbitrarily chosen city is displayed in Fig. B.2. The average Root MSE (RMSE) and Mean Absolute Error (MAE) are summarized in Tab. B.5. The results show that MC-LSTM is able to generalize significantly better than LSTM for this task. The RMSE of MC-LSTM is significantly better than LSTM (p-values $4\text{e}{-10}$, $8\text{e}{-3}$, and $4\text{e}{-10}$ for Istanbul, Berlin, and Moscow, respectively, Wilcoxon test).

B.3. Damped Pendulum

In the area of physics, we consider the problem of modeling a swinging pendulum with friction. The conserved quantity of interest is the total energy. During the movement of the pendulum, kinetic energy is converted into potential energy and vice-versa. Neglecting friction, the total energy is conserved and the movement would continue indefinitely.

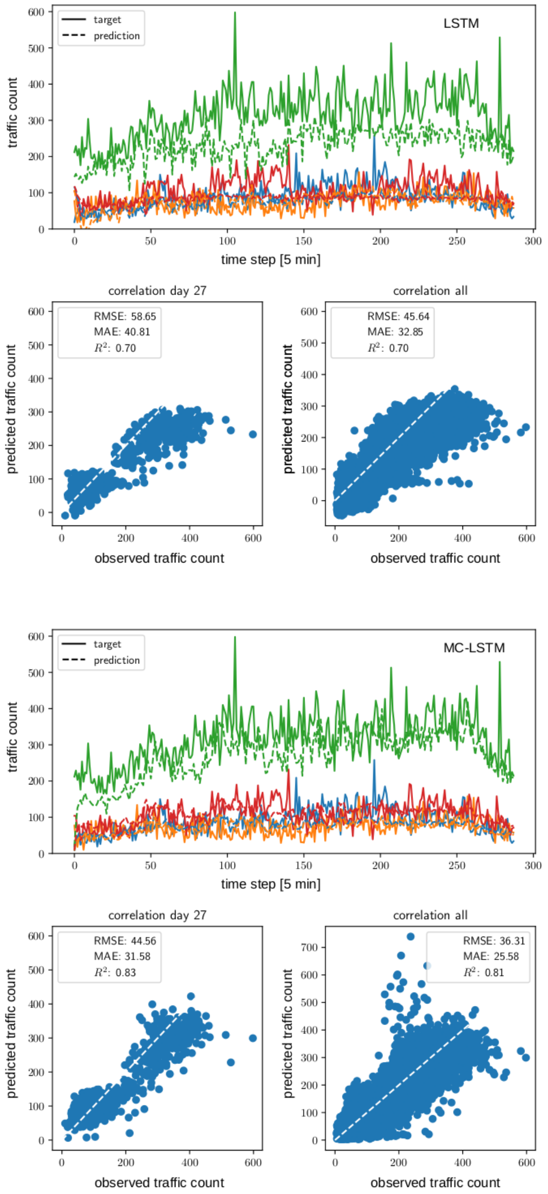

Figure B.2. Traffic forecasting models for outbound traffic in Moscow. An arbitrary day has been chosen for display. Note that both models have only been trained on data at timesteps 71-73, 143-145, and 215-217. Colors indicate the four borders of the frame, i.e., north, east, south and west. **Left:** LSTM predictions shown in dashed lines versus the actual traffic counts (solid lines). **Right:** MC-LSTM predictions shown in dashed lines versus the actual traffic counts (solid lines).

Table B.4. The hyperparameters resulting from the grid search for the traffic forecast experiment.

		hidden	lr	forget bias	initial state	learnable state
Berlin	LSTM	10	0.01	0	–	–
	MC-LSTM	100	0.01	–	0	True
Istanbul	LSTM	100	0.005	5	–	–
	MC-LSTM	50	0.01	–	0	False
Moscow	LSTM	50	0.001	5	–	–
	MC-LSTM	10	0.01	–	0	False

Table B.5. Results on outbound traffic forecast avg RMSE and MAE with 95% confidence intervals over 50 runs

	Istanbul		Berlin		Moscow	
	RMSE	MAE	RMSE	MAE	RMSE	MAE
MC-LSTM	**7.3** ± 0.1	**28** ± 2	**13.6** ± 1.8	**66** ± 1	25.5 ± 1.1	**27.8** ± 1.1
LSTM	142.6 ± 4.4	84 ± 3	135.4 ± 5.0	84 ± 3	45.6 ± 0.8	31.7 ± 0.5

Accounting for friction, energy dissipates and the swinging slows over time until a fixed point is reached. This type of behavior presents a difficulty for machine learning and is impossible for methods that assume the pendulum to be closed systems, such as Hamiltonian Neural Networks (HNNs) (Greydanus et al., 2019). We postulated that both energy conversion and dissipation can be fitted by machine learning models, but that an appropriate inductive bias will allow to generalize from the learned data with more ease.

To train the model, we generated a set of timeseries using the differential equations for a pendulum with friction. For small angles, this problem is equivalent to the harmonic oscillator and an analytic solution exists with which we can compare the models (Iten et al., 2020). We used multiple different settings for initial angle, length of the pendulum, the amount of friction, the length of the training-period and with and without Gaussian noise. Each model received the initial kinetic and potential energy of the pendulum and must predict the consecutive timesteps. The time series starts always with the pendulum at the maximum displacement — i.e., the entire energy in the system is potential energy. We generated timeseries of potential- and kinetic energies by iterating the following settings/conditions: initial amplitude ($\{0.2, 0.4\}$), pendulum length ($\{0.75, 1\}$), length of training sequence in terms of timesteps ($\{100, 200, 400\}$), noise level ($\{0, 0.01\}$), and dampening constant ($\{0.0, 0.1, 0.2, 0.4, 0.8\}$). All combinations of those settings were used to generate a total of 120 datasets, for which we train both models (the autoregressive LSTM and MC-LSTM).

We trained an autoregressive LSTM that receives its current state and a low-dimensional temporal embedding (using nine sinusoidal curves with different frequencies) to predict the potential and kinetic energy of the pendulum. Similarly, MC-LSTM is trained in an autoregressive mode, where a hypernetwork obtains the current state and the same temporal embedding as LSTM. The model-setup is thus similar to an autoregressive model with exogenous variables from classical timeseries modelling literature. To obtain suitable hyperparameters we manually adjusted the learning rate (0.01), hidden size of LSTM (256), the hypernetwork for estimating the redistribution (a fully connected network with 3 layers, ReLU activations and hidden sizes of 50, 100, and 2 respectively), optimizer (Adam, Kingma & Ba, 2015) and the training procedure (crucially, the amount of additionally considered timesteps in the loss after a threshold is reached. See explanation of the used loss below), on a separately generated validation dataset.

For MC-LSTM, a hidden size of two was used so that each state directly maps to the two energies. The hypernetwork consists of three fully connected layers of size 50, 100 and 4, respectively. To account for the critical values at the extreme-points of the pendulum (i.e. the amplitudes — where the energy is present only in the form of potential energy — and the midpoint — where only kinetic energy exists), we slightly offset the cell state from the actual predicted value by using a linear regression with a slope of 1.02 and an intercept -0.01.

For both models, we used a combination of Pearson's correlation of the energy signals and the MSE as a loss function (by subtracting the former mean from the latter). Further, we used a simple curriculum to deal with the long autoregressive nature of the timeseries (Bengio et al., 2015): Starting at a time window of eleven we added five additional timesteps whenever the combined loss was below -0.9.

Overall, MC-LSTM has significantly outperformed LSTM with a mean MSE of 0.01 (standard deviation 0.02) compared to 0.07 (standard deviation 0.14; with a p-value $4.7\text{e}{-10}$, Wilcoxon test).

B.3.1. Qualitative Analysis of the MC-LSTM Models Trained for a Pendulum

In the following, we analyse the behavior of the simplest pendulum setup, i.e., the one without friction. Special to the problem of the pendulum without friction is that there are no mass in- or outputs and the whole dynamic of the system has to be modeled by the redistribution matrix. The initial state of the system is given by the displacement of the pendulum at the start, where all energy is stored as potential energy. Afterwards, the pendulum oscillates, converting potential to kinetic energy and vice-versa.

In MC-LSTM, the conversion between the two forms of energy has to be learned by the redistribution matrix. More specifically, the off-diagonal elements denote the fraction of energy that is converted from one form to the other. In contrast, the diagonal elements of the redistribution matrix denote the fraction of energy that is *not* converted.

In Fig. B.3, we visualize the off-diagonal elements of the redistribution matrix (i.e., the conversion of energy) for the pendulum task without friction, as well as the modeled potential and kinetic energy. We can see that an increasing fraction of energy is converted into the other form, until the total energy of the system is stored as either kinetic or potential energy. As soon as the total energy is e.g. converted into kinetic energy, the corresponding off-diagonal element (the orange line of the upper plot in Fig. B.3) drops to zero. Here, the other off-diagonal element (the blue line of the upper plot in Fig. B.3) starts to increase, meaning that energy is converted back from kinetic into potential energy. Note that the differences in the maximum values of the off-diagonal elements is not important, since at this point the corresponding energy is already approximately zero.

B.3.2. Comparison with Hamiltonian Neural Networks

We aimed at a comparison with HNN in the case of the friction-free pendulum. To this end, we use the data generation process by (Greydanus et al., 2019). We use amplitudes of $\{0.2, 0.3, 0.4, 1\}$, training sequence length $\{100, 200, 400\}$, and noise level $\{0, 0.01\}$, which leads to 24 time-series. We adhere to the HNN reference implementation, which contains a gravity constant of $g = 6$ and mass $m = 0.5$. In the case of the pendulum with friction, the assumptions of HNNs are not met which leads to problematic modeling behavior (see Figure B.4).

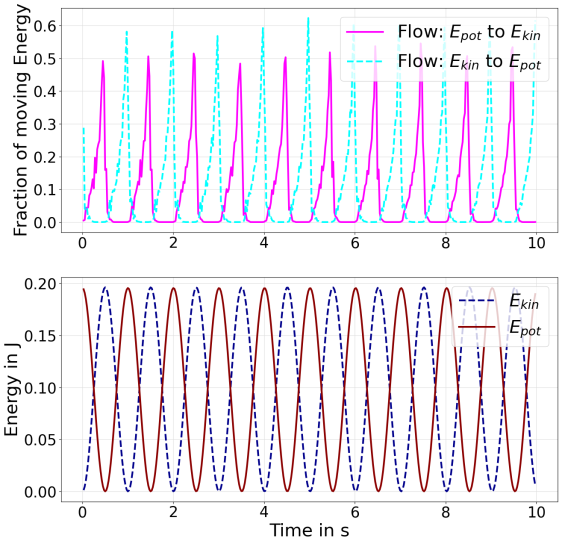

Figure B.3. Redistribution of energies in a pendulum learned by MC-LSTM. The upper plot shows the fraction of energy that is redistributed between the two cells that model E_{pot} and E_{kin} over time. The continuous redistribution of energy results in the two time series of potential and kinetic energy displayed in the lower plot.

The HNNs directly predict the symplectic gradients that provide the dynamics for the pendulum. These gradients can then be integrated to obtain position and momentum for future timesteps. From these prediction, we compute the potential and kinetic energy over time. For MC-LSTM we used the autoregressive version as described above and used position and momentum, both rescaled to amplitude 1, as auxiliary inputs. Note that HNNs are feed-forward networks, and the dynamics are obtained by integrating over their predictions. This implies that due to the periodicity of the data, the samples in the test set could also be in the training data. Moreover, there is only noise on the input data, i.e., position and momentum, but not on the time derivatives, such that HNNs receive non-noisy labels. Therefore the training could be considered less noisy for HNN compared to MC-LSTMs. The mean-squared error of the predictions for the potential and kinetic energy is compared against the analytic solution. Concretely, the average MSE of MC-LSTM is $4.3\text{e}{-4}$, and the MSE of HNNs is $3.0\text{e}{-4}$. On 11 out of 24 datasets, MC-LSTM outperformed HNN, which indicates that there is no significant difference between the two methods (p-value 0.84, binomial test).

B.4. Hydrology

Modeling river discharge from meteorological data (e.g., precipitation, temperature) is one of the most important

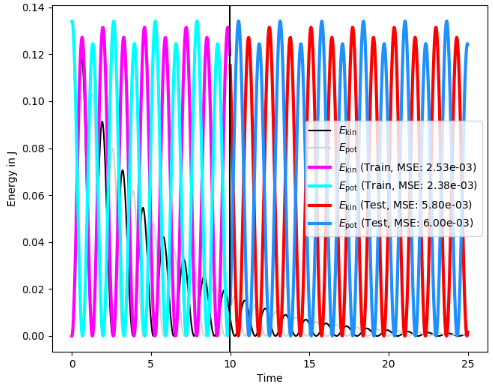

Figure B.4. Example of modeling a pendulum with friction with a HNN. HNNs assume a closed system and cannot model the pendulum with friction, from which energy dissipates.

tasks in hydrology, and is necessary for water resource management and risk mitigation related to flooding. Recently, Kratzert et al. (2019b; 2020) established LSTM-based models as state-of-the-art in rainfall runoff modeling, outperforming traditional hydrological models by a large margin against most metrics (including peak flows, which is critical for flood prediction). However, the hydrology community is still reluctant to adopt these methods (e.g. Beven, 2020). A recent workshop on 'Big Data and the Earth Sciences' (Sellars, 2018) reported that *"[m]any participants who have worked in modeling physical-based systems continue to raise caution about the lack of physical understanding of ML methods that rely on data-driven approaches."*

One of of the most basic principles in watershed modeling is mass conservation. Whether water is treated as a resource (e.g. droughts) or hazard (e.g. floods), a modeller must be sure that they are accounting for all of the water in a catchment. Thus, most models conserve mass (Todini, 1988), and attempt to explicitly implement the most important physical processes. The downside of this 'model everything' strategy is that errors are introduced for every real-world process that is *not* implemented in a model, or implemented incorrectly. In contrast, MC-LSTM is able to learn any necessary behavior that can be induced from the signal (like LSTM) while still conserving the overall water budget.

B.4.1. Details on the Dataset

The data used in all hydrology related experiments is the publicly available Catchment Attributes and Meteorology for Large-sample Studies (CAMELS) dataset (Newman et al., 2014; Addor et al., 2017b). CAMELS contains data for 671 basins and is curated by the US National Center for Atmospheric Research (NCAR). It contains only basins with relatively low anthropogenic influence (e.g., dams and reservoirs) and basin sizes range from 4 to 25 000 km^2. The basins cover a range of different geo- and eco-climatologies, as described by Newman et al. (2015) and Addor et al. (2017a). Out of all 671 basins, we used 447 — these are the basins for which simulations from all benchmark models are available (see Sec. B.4.5). To reiterate, we used benchmark hydrology models that were trained and tested by other groups with experience using these models, and were therefore limited to the 447 basis with results for all benchmark models. The spatial distribution of the 447 basins across the contiguous USA (CONUS) is shown in Fig. B.5.

For each catchment, roughly 30 years of daily meteorological data from three different products exist (DayMet, Maurer, NLDAS). Each meteorological dataset consist of five different variables: daily cumulative precipitation, daily minimum and maximum temperature, average short-wave radiation and vapor pressure. We used the Maurer forcing data because this is the data product that was used by all benchmark models (see Sec. B.4.5). In addition to meteorological data, CAMELS also includes a set of static catchment attributes derived from remote sensing or CONUS-wide available data products. The static catchment attributes can broadly be grouped into climatic, vegetation or hydrological indices, as well as soil and topological properties. In this study, we used the same 27 catchment attributes as Kratzert et al. (2019b). Target data were daily averaged streamflow observations originally from the USGS streamflow gauge network, which are also included in the CAMELS dataset.

Training, validation and test set. Following the calibration and test procedure of the benchmark hydrology models, we trained on streamflow observations from 1 October 1999 through 30 September 2008 and tested on observations from 1 October 1989 to 30 September 1999. The remaining period (1 October 1980 to 30 September 1989) was used as validation period for hyperparameter tuning.

B.4.2. Details on the Training Setup and MC-LSTM Hyperparameters

The general model setup follows insights from previous studies (Kratzert et al., 2018; 2019b;a; 2020), where LSTMs were used for the same task. We use sequences of 365 timesteps (days) of meteorological inputs to predict discharge at the last timestep of the sequence (sequence-to-one prediction). The mass input x in this experiment was catchment averaged precipitation (mm/day) and the auxiliary inputs $\boldsymbol{a}$ were the 4 remaining meteorological variables (min. and max. temperature, short-wave radiation and vapor pressure) as well as the 27 static catchment attributes, which are constant over time.

We tested a variety of MC-LSTM model configurations and adaptions for this specific task, which are briefly described

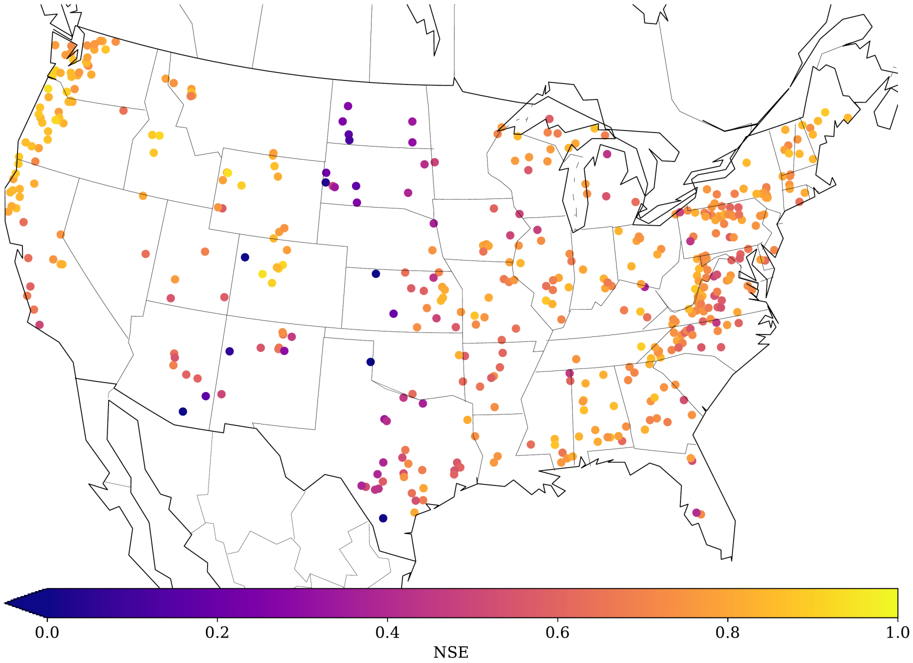

Figure B.5. Spatial distribution of the 447 catchments considered in this study. The color denotes the Nash-Sutcliffe Efficiency of the MC-LSTM ensemble for each basin, where a value of 1 means perfect predictions.

below:

1. **Processing auxiliary inputs with LSTM**: Instead of directly using the auxiliary inputs in the input gate (Eq. 5), output gate (Eq. 6) and time-dependent mass redistribution (Eq. 8), we first processed the auxiliary inputs a with LSTM and then used the output of this LSTM as the auxiliary inputs. The idea was to add additional memory for the auxiliary inputs, since in its base form only mass can be stored in the cell states of MC-LSTM. This could be seen as a specific adaption for the rainfall runoff modeling application, since information about the weather today and in the past ought to be useful for controlling the gates and mass redistribution. Empirically however, we could not see any significant performance gain and therefore decided to not use the more complex version with an additional LSTM.

2. **Auxiliary output + regularization to account for evapotranspiration**: Of all precipitation falling in a catchment, only a part ends as discharge in the river. Large portions of precipitation are lost to the atmosphere in form of evaporation (from e.g. open water surfaces) and transpiration (from e.g. plants and trees), and to groundwater. One approach to account for this "mass loss" is the following: instead of summing over outgoing mass (Eq. 4), we used a linear layer to connect the outgoing mass to two output neurons. One neuron was fitted against the observed discharge data, while the second was used to estimate water loss due to unobserved sinks. A regularization term was added to the loss function to account for this. This regularization term was computed as the difference between the sum of the outgoing mass from MC-LSTM and the sum over the two output neurons. This did work, and the timeseries of the second auxiliary output neuron gave interesting results (i.e. matching the expected behavior of the annual evapotranspiration cycle), however results were not significantly better compared to our final model setup, which is why we rejected this architectural change.

3. **Explicit trash cell** Another way to account for evapotranspiration that we tested is to allow the model to use one memory cell as explicit "trash cell". That is, instead of deriving the final model prediction as the sum over the *entire* outgoing mass vector, we only calculate the sum over all but e.g. one element (see Eq. 4). This simple modification allows the model to use e.g. the first memory cell to discard mass from the system, which is then ignored for the model prediction. We found that this modification improved performance, and thus integrated it into our final model setup.

4. **Input/output scaling to account for input/output uncertainty**: Both, input and output data in our ap-

plications inherit large uncertainties (Nearing et al., 2016), which is not ideal for mass-conserving models (and likely one of the reasons why LSTM performs so well compared to all other mass-conserving models). To account for that, we tried three different adaptions. First, we used a small fully connected network to derive time-dependent scaling weights for the mass input, which we regularized to be close to one. Second, we used a linear layer with positive weights to map the outgoing mass to the final model prediction, where all weights were initialized to one and the bias to zero. Third, we combined both. Out of the three, the input scaling resulted in the best performing model, however the results were worse than not scaling.

5. **Time-dependent redistribution matrix variants**: For this experiment, a time-dependent redistribution matrix is necessary, since the underlying real-world processes (such as snow melt and thus conversion from snow into e.g. soil moisture or surface runoff) are time-dependent. Since using the redistribution matrix as proposed in Eq. 8 is memory-demanding, especially for models with larger numbers of memory cells, we also tried to use a different method for this experiment. Here, we learned a fixed matrix (as in Eq. 7) and only calculated two vectors for each timestep. The final redistribution matrix was then derived as the outer product of the two time-dependent vectors and the static matrix. This resulted in lower memory consumption, however the model performance deteriorated significantly, which could be a hint toward the complexity required to learn the redistributing processes in this problem.

6. **Activation function of the redistribution matrix**: We tested several different activation functions for the redistribution matrix in this experiment. Among those were the normalized sigmoid function, the softmax function (as in Eq. 8) and the normalized ReLU activation function (see Eq. 9). We could achieve the best results using the normalized ReLU variant and can only hypothesize the reason for that: In this application (rainfall-runoff modelling) there are several state processes that are strictly disconnected. One example is snow and groundwater: groundwater will never turn into snow and snow will never transform into groundwater (not directly at least, it will first need to percolate through upper soil layers). Using normalized sigmoids or softmax makes it numerically harder (or impossible) to not distributed at least *some* mass between every cell — because activations can never be exactly zero. The normalized ReLU activation can do so, however, which might be the reason that it worked better in this case.

7. **Activation function of the input gate**: Similar to the redistribution matrix, different activation functions can be used for the input gate. We tested the same three functions as for the redistribution matrix. For the input gate, the normalized sigmoid function resulted in the best performing model which was therefore used.

As an extension to the standard MC-LSTM model introduced in Eq. (5) to Eq. (8), we also used the mass input (precipitation) in all gates. The reason is the following: Different amounts of precipitations can lead to different processes. For example, low amounts of precipitation could be absorbed by the soil and stored as soil moisture, leading to effectively no immediate discharge contribution. Large amounts of precipitation on the other hand, could lead to direct surface runoff, if the water cannot infiltrate the soil at the rate of the precipitation falling down. Therefore, it is crucial that the gates have access to the information contained in the precipitation input. The final model design used in all hydrology experiments is described by the following equations:

$$\boldsymbol{m}_{\text{tot}}^t = \boldsymbol{R}^t \cdot \boldsymbol{c}^{t-1} + \boldsymbol{i}^t \cdot x^t \tag{1}$$

$$\boldsymbol{c}^t = (\boldsymbol{1} - \boldsymbol{o}^t) \odot \boldsymbol{m}_{\text{tot}}^t \tag{2}$$

$$\boldsymbol{h}^t = \boldsymbol{o}^t \odot \boldsymbol{m}_{\text{tot}}^t \tag{3}$$

$$\widehat{y} = \sum_{i=2}^{n} h_i^t, \tag{4}$$

with the gates being defined by

$$\boldsymbol{i}^t = \tilde{\sigma}(\boldsymbol{W}_{\text{i}} \cdot \boldsymbol{a}^t + \boldsymbol{U}_{\text{i}} \cdot \frac{\boldsymbol{c}^{t-1}}{\|\boldsymbol{c}^{t-1}\|_1} + \boldsymbol{V}_{\text{i}} \cdot x^t + \boldsymbol{b}_{\text{i}}) \tag{5}$$

$$\boldsymbol{o}^t = \sigma(\boldsymbol{W}_{\text{o}} \cdot \boldsymbol{a}^t + \boldsymbol{U}_{\text{o}} \cdot \frac{\boldsymbol{c}^{t-1}}{\|\boldsymbol{c}^{t-1}\|_1} + \boldsymbol{V}_{\text{o}} \cdot x^t + \boldsymbol{b}_{\text{o}}) \tag{6}$$

$$\boldsymbol{R}^t = \widetilde{\text{ReLU}}\left(\mathbf{W}_{\text{r}} \cdot \boldsymbol{a}^t + \mathbf{U}_{\text{r}} \cdot \frac{\boldsymbol{c}^{t-1}}{\|\boldsymbol{c}^{t-1}\|_1} + \mathbf{V}_{\text{r}} \cdot x^t + \boldsymbol{B}_{\text{r}}\right), \tag{7}$$

where $\tilde{\sigma}$ is the *normalized logistic function* and $\widetilde{\text{ReLU}}$ is the *normalized rectified linear unit* (ReLU) that we define in the following. The normalized logistic function defined of the input gate is defined by:

$$\tilde{\sigma}(i_k) = \frac{\sigma(i_k)}{\sum_k \sigma(i_k)}. \tag{8}$$

In this experiment, the activation function for the redistribution gate is the normalized ReLU function defined by:

$$\widetilde{\mathrm{ReLU}}(s_k) = \frac{\max(s_k, 0)}{\sum_k \max(s_k, 0)}, \tag{9}$$

where $\boldsymbol{s}$ is some input vector to the normalized ReLU function.

We manually tried different sets of hyperparameters, because a large-scale automatic hyperparameter search was not feasible. Besides trying out all variants as described above, the main hyperparameter that we tuned for the final model was the number of memory cells. For other parameters, such as learning rate, mini-batch size, number of training epochs, we relied on previous work using LSTMs on the same dataset.

The final hyperparameters are a hidden size of 64 memory cells and a mini-batch size of 256. We used the Adam optimizer (Kingma & Ba, 2015) with a scheduled learning rate starting at 0.01 then lowering the learning rate after 20 epochs to 0.005 and after another 5 epochs to 0.001. We trained the model for a total number of 30 epochs and used the weights of the last epoch for the final model evaluation. All weight matrices were initialized as (semi) orthogonal matrices (Saxe et al., 2014) and all bias terms with a constant value of zero. The only exception was the bias of the output gate, which we initialized to -3, to keep the output gate closed at the beginning of the training.

B.4.3. Details on the Evaluation Metrics

Table B.6 lists the definition of all metrics used in the hydrology experiments as well as the corresponding references.

B.4.4. Details on the LSTM Model

For LSTM, we largely relied on expertise from previous studies (Kratzert et al., 2018; 2019b;a; 2020). The only hyperparameter we adapted was the number of memory cells, since we used fewer basins (447) than in the previous studies (531). We found that LSTM with 128 memory cells, compared to the 256 used in previous studies, resulted in slightly better results. Apart from that, we trained LSTMs with the same inputs and settings (sequence-to-one with a sequence length of 365) as described in the previous section for MC-LSTM. We used the standard LSTM implementation from the PyTorch package (Paszke et al., 2019), i.e., with forget gate (Gers et al., 2000). We manually initialized the bias of the forget gate to be 3 in order to keep the forget gate open at the beginning of the training.

B.4.5. Details on the Benchmark Models

The benchmark models were first collected by Kratzert et al. (2019b). All models were configured, trained and run by several different research groups, most often the respective model developers themselves. This was done to avoid any potential to favor our own models. All models used the same forcing data (Maurer) and the same time periods to train and test. The models can be classified in two groups:

1. *Models trained for individual watersheds*. These are SAC-SMA (Newman et al., 2017), VIC (Newman et al., 2017), three different model structures of FUSE[1], mHM (Mizukami et al., 2019) and HBV (Seibert et al., 2018). For the HBV model, two different simulations exist: First, the ensemble average of 1000 untrained HBV models (lower benchmark) and second, the ensemble average of 100 trained HBV models (upper benchmarks). For details see (Seibert et al., 2018).
2. *Models trained regionally*. For hydrological models, regional training means that one parameter transfer model was trained, which estimates watershed-specific model parameters through globally trained model functions from e.g. soil maps or other catchment attributes. For this setting, the benchmark dataset includes simulations of the VIC model (Mizukami et al., 2017) and mHM (Rakovec et al., 2019).

B.4.6. Detailed Results

Table B.7 provides results for MC-LSTM and LSTM averaged over the $n = 10$ model repetitions.

B.5. Ablation Study

In order to demonstrate that the design choices of MC-LSTM are necessary together to enable accurate predictive models, we performed an ablation study. In this study, we make the following changes to the input gate, the redistribution operation, and the output gate, to test if mass conservation in the individual parts is necessary.

1. Input gate: We change the activation function of the input gate from a normalized sigmoid function to the standard sigmoid function, thus resulting in the input gate of a standard LSTM. Since the sigmoid function is bounded to $(0, 1)$, the mass input x at every timestep t that is added into the system can be scaled between $(0, n * x^t)$.
2. Redistribution matrix: We remove the normalized activation function from the redistribution matrix and instead use a linear activation function. This allows for unconstrained and scaled flow of mass from each memory cell into each other memory cell.
3. Output gate: Instead of removing the outgoing mass $(\boldsymbol{o}^t \odot \boldsymbol{m}^t_{\mathrm{tot}})$ from the cell states at each timestep t,

[1] Provided by Nans Addor on personal communication

Table B.6. Definition of all metrics used in the hydrology experiments. The NSE is defined as the R^2 between simulated, $\hat{y}$, and observed, y, runoff and is listed for completion. FHV and FLV are both derived from the flow duration curve, which is a cumulative frequency curve of the discharge. H for the FHV and L for the FLV correspond to the 2% highest flow and the 30% lowest flow, respectively.

Metric	Reference	Equation
Nash-Sutcliff-Efficiency (NSE)[a]	Nash & Sutcliffe (1970)	$1 - \frac{\sum_{t=1}^{T}(\widehat{y}^t - y^t)^2}{\sum_{t=1}^{T}(y^t - \bar{y})^2}$
β-NSE Decomposition[b]	Gupta et al. (2009)	$(\mu_{\widehat{y}} - \mu_y)/\sigma_y$
Top 2% peak flow bias (FHV)[c]	Yilmaz et al. (2008)	$\frac{\sum_{h=1}^{H}(\widehat{y}_h - y_h)}{\sum_{h=1}^{H} y_h} \times 100$
30% low flow bias (FLV)[d]	Yilmaz et al. (2008)	$\frac{\sum_{l=1}^{L}(\log(\widehat{y}_l) - \log(\widehat{y}_L)) - \sum_{l=1}^{L}(\log(y_l) - \log(y_L))}{\sum_{l=1}^{L}(\log(y_l) - \log(y_L))} \times 100$

[a]: *Nash-Sutcliffe efficiency:* $(-\infty, 1]$, *values closer to one are desirable.*
[b]: β*-NSE decomposition:* $(-\infty, \infty)$, *values closer to zero are desirable.*
[c]: *Top 2% peak flow bias:* $(-\infty, \infty)$, *values closer to zero are desirable.*
[d]: *Bottom 30% low flow bias:* $(-\infty, \infty)$, *values closer to zero are desirable.*

Table B.7. Model robustness of MC-LSTM and LSTM results over the $n = 10$ different random seeds. For all $n = 10$ models, we calculated the median performance for each metric and report the mean and standard deviation of the median values in this table.

	MC[a]	NSE[b]	β-NSE[c]	FLV[d]	FHV[e]
MC-LSTM Single	✓	0.726±0.003	-0.021±0.003	-38.7±3.2	-13.9±0.7
LSTM Single	✗	0.737±0.003	-0.035±0.005	13.6±3.4	-14.8±1.0

[a]: *Mass conservation (MC).*
[b]: *Nash-Sutcliffe efficiency:* $(-\infty, 1]$, *values closer to one are desirable.*
[c]: β*-NSE decomposition:* $(-\infty, \infty)$, *values closer to zero are desirable.*
[d]: *Bottom 30% low flow bias:* $(-\infty, \infty)$, *values closer to zero are desirable.*
[e]: *Top 2% peak flow bias:* $(-\infty, \infty)$, *values closer to zero are desirable.*

Table B.8. Ablation study results of the hydrology experiment. Models are trained for five, random basin with nine model repetitions. We computed the median over the repetitions and then the mean over the five basins.

	MC[a]	NSE[b]
MC-LSTM	✓	0.635 ± 0.102
MC-LSTM w. softmax	✓	0.650 ± 0.095
MC-LSTM − input	✗	0.603 ± 0.123
MC-LSTM − output	✗	0.55 ± 0.097
MC-LSTM − redis.[c]	✗	-4.229 ± 8.982

[a]: *Mass conservation (MC).*
[b]: *Nash-Sutcliffe efficiency:* $(-\infty, 1]$, *values closer to one are desirable.*
[c]: *For one out of five basins, all nine model repetitions resulted into NaNs during training. Here, we report the statistics calculated from only the four successful basins.*

we leave the cell states unchanged and keep all mass within the system.

We test these variants on data from the hydrology experiment. We chose 5 random basins to limit computational expenses and trained nine repetitions for each configuration and basin. The results are compared against the full mass-conserving MC-LSTM architecture as described in App. B.4.2 and reported in Table B.8. The results of the ablation study indicate that the design of the input gate, redistribution matrix, and output gate, are necessary together for proficient predictive performance. The strongest decrease in performance is observed if redistribution matrix does not conserve mass, and smaller decreases if input or output gate do not conserve mass. We also tested a variant, where we used the softmax activation function in the input gate, instead of the normalized sigmoid that was used in the hydrology experiments (see Sec. B.4.2). Both mass conserving variants, once with normalized sigmoid as activation function and once with softmax, achieve similar performance, while the variant with softmax is slightly better. However, as stated in Sec. B.4.2, we also tested this variant on the multi-basin version that we trained for the hydrology experiments. Here, the normalized sigmoid activation function resulted in better model performance. This emphasizes that both variants are viable options and the exact design of the MC-LSTM might be task dependent.

B.6. Runtime

Section 3 provides a comparison of MC-LSTM and LSTM in terms of computational complexity. Since this comparison is rather abstract, we also conducted an empirical evaluation of the runtime. The empirical runtimes of the forward pass for a single batch for both MC-LSTM and LSTM are listed in table B.9. Note that the backward pass should scale similarly to the forward pass.

	CPU	GPU
MC-LSTM	951^{+1}_{-5}	236^{+16}_{-1}
LSTM	205^{+8}_{-10}	121^{+0}_{-0}

Table B.9. Median runtime in ms of 5 forward passes with indication of 25 and 75% quantiles. Timings were executed on a PC with AMD Ryzen 7 2700 CPU and Nvidia GTX 1070Ti GPU.

We used the prototypical architecture for the hydrology experiments. Concretely, both models received 1 mass input, 30 auxiliary inputs and had 64 hidden units. A batch of 256 sequences was used, where each sequence has 365 timesteps. To keep the comparison fair, we used a custom LSTM rather than the highly optimised default implementation that is available in pytorch (Paszke et al., 2019).

C. Theorems & Proofs

Theorem 1 (Conservation property)**.** *Let* $m_c^\tau = \sum_k c_k^\tau$ *and* $m_h^\tau = \sum_k h_k^\tau$ *be, respectively, the* mass *in the MC-LSTM storage and the outputs at time* τ*. At any timestep* τ*, we have:*

$$m_c^\tau = m_c^0 + \sum_{t=1}^{\tau} x^t - \sum_{t=1}^{\tau} m_h^t.$$

That is, the change of mass in the cell states is the difference between input and output mass, accumulated over time.

Proof. The proof is by induction and we use $\boldsymbol{m}_{\text{tot}} = \boldsymbol{R}^t \cdot \boldsymbol{c}^{t-1} + \boldsymbol{i}^t \cdot x^t$ from Eq.(2).

For $\tau = 0$, we have $m_c^0 = m_c^0 + \sum_{t=1}^{0} x^t - \sum_{t=1}^{0} m_h^t$, which is trivially true when using the convention that $\sum_{t=1}^{0} = 0$.

Assuming that the statement holds for $\tau = T$, we show that it must also hold for $\tau = T + 1$.

Starting from Eq. (3), the mass of the cell states at time $T + 1$ is given by:

$$m_c^{T+1} = \sum_{k=1}^{K} (1 - o_k) m_{\text{tot},k}^{T+1} = \sum_{k=1}^{K} m_{\text{tot},k}^{T+1} - \sum_{k=1}^{K} o_k m_{\text{tot},k}^{T+1},$$

where $m_{\text{tot},k}^t$ is the k-th entry of the result from Eq. (2) (at timestep t). The sum over entries in the first term can be simplified as follows:

$$\begin{aligned}\sum_{k=1}^{K} m_{\text{tot},k}^{T+1} &= \sum_{k=1}^{K} \left(\sum_{j=1}^{K} r_{kj} c_j^T + i_k x^{T+1} \right) \\ &= \sum_{j=1}^{K} c_j^T \left(\sum_{k=1}^{K} r_{kj} \right) + x^{T+1} \sum_{k=1}^{K} i_k \\ &= m_c^T + x^{T+1}.\end{aligned}$$

The final simplification is possible because $\boldsymbol{R}$ and $\boldsymbol{i}$ are (left-)stochastic. The mass of the outputs can then be computed from Eq. (4):

$$m_h^{T+1} = \sum_{k=1}^{K} o_k m_{\text{tot},k}^{T+1}.$$

Putting everything together, we find

$$\begin{aligned}m_c^{T+1} &= \sum_{k=1}^{K} m_{\text{tot},k}^{T+1} - \sum_{k=1}^{K} o_k m_{\text{tot},k}^{T+1} \\ &= m_c^T + x^{T+1} - m_h^{T+1} \\ &= m_c^0 + \sum_{t=1}^{T} x^t - \sum_{t=1}^{T} m_h^t + x^{T+1} - m_h^{T+1} \\ &= m_c^0 + \sum_{t=1}^{T+1} x^t - \sum_{t=1}^{T+1} m_h^t\end{aligned}$$

By the principle of induction, we conclude that mass is conserved, as specified in Eq. (9). □

Corollary 1. *In each timestep τ, the cell states c_k^τ are bounded by the sum of mass inputs $\sum_{t=1}^{\tau} x^\tau + m_c^0$, that is $|c_k^\tau| \leq \sum_{t=1}^{\tau} x^\tau + m_c^0$. Furthermore, if the series of mass inputs converges $\lim_{\tau \to \infty} \sum_{t=1}^{\tau} x^\tau = m_x^\infty$, then also the sum of cell states converges.*

Proof. Since $c_k^t \geq 0$, $x^t \geq 0$ and $m_h^t \geq 0$ for all k and t,

$$|c_k^\tau| = c_k^\tau \leq \sum_{k=1}^{K} c_k^\tau = m_c^\tau \leq \sum_{t=1}^{\tau} x^\tau + m_c^0, \qquad (10)$$

where we used Theorem 1. Convergence follows immediately through the *comparison test*. □

D. On Random Markov Matrices.

When initializing an MC-LSTM model, the entries of the redistribution matrix $\boldsymbol{R}$ of dimension $K \times K$ are created from non-negative and iid random variables $(s_{ij})_{1 \leq i,j \leq K}$ with finite means m and variances σ^2 and bounded fourth moments. We collect them in a matrix $\boldsymbol{S}$. Next we assume that those entries get column-normalized to obtain the random Markov matrix $\boldsymbol{R}$.

Properties of Markov matrices and random Markov matrices. Let $\lambda_1, \ldots, \lambda_K$ be the eigenvalues and $s_1, \ldots, s_K$ be the singular values of $\boldsymbol{R}$, ordered such that $|\lambda_1| \geq \ldots \geq |\lambda_K|$ and $s_1 \geq \ldots \geq s_k$. We then have the following properties for any Markov matrix (not necessarily random):

- $\lambda_1 = 1$.
- $\boldsymbol{1}^T \boldsymbol{R} = \boldsymbol{1}^T$.
- $s_1 = \|\boldsymbol{R}\|_2 \leq \sqrt{K}$.

Furthermore, for random Markov matrices, we have

- $\lim_{K \to \infty} s_1 = 1$ (Bordenave et al., 2012, Theorem 1.2)

For the reader's convenience we briefly discuss further selected interesting properties of random Markov matrices in the next paragraph, especially concerning the global behavior of their eigenvalues and singular values.

Circular and Quartercircular law for random Markov matrices. In random matrix theory one major field of interest concerns the behavior of eigenvalues and singular values when $K \to \infty$. One would like to find out how the limiting distribution of the eigenvalues or singular values looks like. To discuss the most important results in this direction for large Markov matrices $\boldsymbol{R}$, let us introduce some notation.

- δ_a denotes the Dirac delta measure centered at a.
- By $\mu_{\boldsymbol{R}} = \frac{1}{K} \sum_{k=1}^{K} \delta_{\lambda_k}$ we denote the empirical spectral density of the eigenvalues of $\boldsymbol{R}$.
- Similarly we define the empirical spectral density of the singular values of $\boldsymbol{R}$ as: $\nu_{\boldsymbol{R}} = \frac{1}{K} \sum_{k=1}^{K} \delta_{s_k}$.
- $\mathcal{Q}_\sigma$ denotes the quartercircular distribution on the interval $[0, \sigma]$ and
- $\mathcal{U}_\sigma$ the uniform distribution on the disk $\{z \in \mathbb{C} : |z| \leq \sigma\}$.

Then we have as $K \to \infty$:

- **Quarter cirular law theorem:** (Bordenave et al., 2012, Theorem 1.1): $\nu_{\sqrt{K}\boldsymbol{R}} \to \mathcal{Q}_\sigma$ almost surely.
- **Cirular law theorem:** (Bordenave et al., 2012, Theorem 1.3): $\nu_{\sqrt{K}\boldsymbol{R}} \to \mathcal{U}_\sigma$ almost surely.

The convergence here is understood in the sense of weak convergence of probability measures with respect to bounded continuous functions. Note that those two famous theorems originally appeared for $\frac{1}{\sqrt{K}}\boldsymbol{S}$ instead of $\sqrt{K}\boldsymbol{R}$. Of course much more details on those results can be found in Bordenave et al. (2012).

Gradient flow of MC-LSTM for random redistributions. Here we provide a short note on the gradient dynamics of the cell state in a random MC-LSTM, hence, at initialization of the model. Specifically we want to provide some heuristics based on the arguments about the behavior of large stochastic matrices. Let us start by recalling the formula for $\boldsymbol{c}^t$:

$$\boldsymbol{c}^t = (\boldsymbol{1} - \boldsymbol{o}^t) \odot (\boldsymbol{R}^t \cdot \boldsymbol{c}^{t-1} + \boldsymbol{i}^t \cdot x^t). \tag{11}$$

Now we investigate the gradient of $\| \frac{\partial \boldsymbol{c}^t}{\partial \boldsymbol{c}^{t-1}} \|_2$ in the limit $K \to \infty$. We assume that for $K \to \infty$, $\boldsymbol{o}^t \approx \boldsymbol{0}$ and $\boldsymbol{i}^t \approx \boldsymbol{0}$ for all t. Thus we approximately have:

$$\| \frac{\partial \boldsymbol{c}^t}{\partial \boldsymbol{c}^{t-1}} \|_2 \approx \| \boldsymbol{R}^t \|_2. \tag{12}$$

$\boldsymbol{R}^t$ is a stochastic matrix, and $s_1 = \|\boldsymbol{R}^t\|_2$ is its largest singular value. Theorem 1.2 from Bordenave et al. (2012) ensures that $\|\boldsymbol{R}^t\|_2 = 1$ for $K \to \infty$ under reasonable moment assumptions on the distribution of the unnormalized entries (see above). Thus we are able to conclude $\| \frac{\partial \boldsymbol{c}^t}{\partial \boldsymbol{c}^{t-1}} \|_2 \approx 1$ for large K and all t, which can prevent the gradients from exploding.

References

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

Addor, N., Newman, A. J., Mizukami, N., and Clark, M. P. Catchment attributes for large-sample studies. *Boulder, CO: UCAR/NCAR*, 2017b.

Arjovsky, M., Shah, A., and Bengio, Y. Unitary evolution recurrent neural networks. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, pp. 1120–1128. PMLR, 2016.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 28, pp. 1171–1179. Curran Associates, Inc., 2015.

Beven, K. Deep learning, hydrological processes and the uniqueness of place. *Hydrological Processes*, 34(16): 3608–3613, 2020.

Bordenave, C., Caputo, P., and Chafai, D. Circular law theorem for random markov matrices. *Probability Theory and Related Fields*, 152(3-4):751–779, 2012.

Cui, Z., Henrickson, K., Ke, R., and Wang, Y. Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

Dehaene, S. *The number sense: How the mind creates mathematics*. Oxford University Press, 2 edition, 2011. ISBN 9780199753871.

Gallistel, C. R. Finding numbers in the brain. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 373(1740), 2018. doi: 10.1098/rstb.2017.0119.

Gers, F. A., Schmidhuber, J., and Cummins, F. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, 2000.

Greydanus, S., Dzamba, M., and Yosinski, J. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, volume 32, pp. 15353–15363. Curran Associates, Inc., 2019.

Gupta, H. V., Kling, H., Yilmaz, K. K., and Martinez, G. F. Decomposition of the mean squared error and nse performance criteria: Implications for improving hydrological modelling. *Journal of hydrology*, 377(1-2):80–91, 2009.

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

Iten, R., Metger, T., Wilming, H., Del Rio, L., and Renner, R. Discovering physical concepts with neural networks. *Physical Review Letters*, 124(1):010508, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kratzert, F., Klotz, D., Brenner, C., Schulz, K., and Herrnegger, M. Rainfall–runoff modelling using long short-term memory (lstm) networks. *Hydrology and Earth System Sciences*, 22(11):6005–6022, 2018.

Kratzert, F., Klotz, D., Herrnegger, M., Sampson, A. K., Hochreiter, S., and Nearing, G. S. Toward improved predictions in ungauged basins: Exploiting the power of machine learning. *Water Resources Research*, 55(12): 11344–11354, 2019a.

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

Kratzert, F., Klotz, D., Hochreiter, S., and Nearing, G. A note on leveraging synergy in multiple meteorological datasets with deep learning for rainfall-runoff modeling. *Hydrology and Earth System Sciences Discussions*, 2020: 1–26, 2020.

Kreil, D. P., Kopp, M. K., Jonietz, D., Neun, M., Gruca, A., Herruzo, P., Martin, H., Soleymani, A., and Hochreiter, S. The surprising efficiency of framing geo-spatial time series forecasting as a video prediction task–insights from the iarai traffic4cast competition at neurips 2019. In *NeurIPS 2019 Competition and Demonstration Track*, pp. 232–241. PMLR, 2020.

Liu, Y., Liu, Z., and Jia, R. Deeppf: A deep learning based architecture for metro passenger flow prediction. *Transportation Research Part C: Emerging Technologies*, 101:18–34, 2019.

Madsen, A. and Johansen, A. R. Neural arithmetic units. In *International Conference on Learning Representations*, 2020.

Mizukami, N., Clark, M. P., Newman, A. J., Wood, A. W., Gutmann, E. D., Nijssen, B., Rakovec, O., and Samaniego, L. Towards seamless large-domain parameter estimation for hydrologic models. *Water Resources Research*, 53(9):8020–8040, 2017.

Mizukami, N., Rakovec, O., Newman, A. J., Clark, M. P., Wood, A. W., Gupta, H. V., and Kumar, R. On the choice of calibration metrics for "high-flow" estimation using hydrologic models. *Hydrology and Earth System Sciences*, 23(6):2601–2614, 2019.

Nash, J. E. and Sutcliffe, J. V. River flow forecasting through conceptual models part i—a discussion of principles. *Journal of hydrology*, 10(3):282–290, 1970.

Nearing, G. S., Tian, Y., Gupta, H. V., Clark, M. P., Harrison, K. W., and Weijs, S. V. A philosophical basis for hydrological uncertainty. *Hydrological Sciences Journal*, 61(9):1666–1678, 2016.

Newman, A., Sampson, K., Clark, M., Bock, A., Viger, R., and Blodgett, D. A large-sample watershed-scale hydrometeorological dataset for the contiguous USA. *Boulder, CO: UCAR/NCAR*, 2014.

Newman, A., Clark, M., Sampson, K., Wood, A., Hay, L., Bock, A., Viger, R., Blodgett, D., Brekke, L., Arnold, J., et al. Development of a large-sample watershed-scale hydrometeorological data set for the contiguous USA: data set characteristics and assessment of regional variability in hydrologic model performance. *Hydrology and Earth System Sciences*, 19(1):209–223, 2015.

Newman, A. J., Mizukami, N., Clark, M. P., Wood, A. W., Nijssen, B., and Nearing, G. Benchmarking of a physically based hydrologic model. *Journal of Hydrometeorology*, 18(8):2215–2225, 2017.

Nieder, A. The neuronal code for number. *Nature Reviews Neuroscience*, 17(6):366–382, 2016. doi: https://doi.org/10.1038/nrn.2016.40.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pp. 8024–8035. Curran Associates, Inc., 2019.

Rakovec, O., Mizukami, N., Kumar, R., Newman, A. J., Thober, S., Wood, A. W., Clark, M. P., and Samaniego, L. Diagnostic evaluation of large-domain hydrologic models calibrated across the contiguous united states. *Journal of Geophysical Research: Atmospheres*, 124(24):13991–14007, 2019.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations*, 2014.

Seibert, J., Vis, M. J. P., Lewis, E., and van Meerveld, H. J. Upper and lower benchmarks in hydrological modelling. *Hydrological Processes*, 32(8):1120–1125, 2018.

Sellars, S. "grand challenges" in big data and the earth sciences. *Bulletin of the American Meteorological Society*, 99(6):ES95–ES98, 2018.

Tedjopurnomo, D. A., Bao, Z., Zheng, B., Choudhury, F., and Qin, A. A survey on modern deep neural network for traffic prediction: Trends, methods and challenges. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

Todini, E. Rainfall-runoff modeling — past, present and future. *Journal of Hydrology*, 100(1):341–352, 1988. ISSN 0022-1694.

Trask, A., Hill, F., Reed, S. E., Rae, J., Dyer, C., and Blunsom, P. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*, volume 31, pp. 8035–8044. Curran Associates, Inc., 2018.

Vanajakshi, L. and Rilett, L. Loop detector data diagnostics based on conservation-of-vehicles principle. *Transportation research record*, 1870(1):162–169, 2004.

Yilmaz, K. K., Gupta, H. V., and Wagener, T. A process-based diagnostic approach to model evaluation: Application to the nws distributed hydrologic model. *Water Resources Research*, 44(9):1–18, 2008. ISSN 00431397.

Zhao, Z., Chen, W., Wu, X., Chen, P. C., and Liu, J. Lstm network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2):68–75, 2017.