GATING REVISITED: DEEP MULTI-LAYER RNNS THAT CAN BE TRAINED

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent Neural Networks (RNNs) are widely used models for sequence data. Just like for feedforward networks, it has become common to build "deep" RNNs, i.e., stack multiple recurrent layers to obtain higher-level abstractions of the data. However, this works only for a handful of layers. Unlike feedforward networks, stacking more than a few recurrent units (e.g., LSTM cells) usually hurts model performance, the reason being vanishing or exploding gradients during training. We investigate the training of multi-layer RNNs and examine the magnitude of the gradients as they propagate through the network. We show that, depending on the structure of the basic recurrent unit, the gradients are systematically attenuated or amplified, so that with an increasing depth they tend to vanish, respectively explode. Based on our analysis we design a new type of gated cell that better preserves gradient magnitude, and therefore makes it possible to train deeper RNNs. We experimentally validate our design with five different sequence modelling tasks on three different datasets. The proposed stackable recurrent (STAR) cell allows for substantially deeper recurrent architectures, with improved performance.

1 INTRODUCTION

Recurrent Neural Networks (RNN) have established themselves as a powerful tool for modelling sequential data. They have significantly advanced a number of applications, notably language processing and speech recognition (Sutskever et al., 2014; Graves et al., 2013; Vinyals & Le, 2015).

The basic building block of an RNN is a computational *unit* (or *cell*) that combines two inputs: the data of the current time step in the sequence and the unit's own output from the previous time step. While RNNs are an effective approach that can in principle handle sequences of arbitrary and varying length, they are (in their basic form) challenged by long-term dependencies, since learning those would require the propagation of gradients over many time steps. To alleviate this limitation, *gated* architectures have been proposed, most prominently Long Short-Term Memory (LSTM) cells (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Units (GRU, Chung et al., 2014). They use a gating mechanism to store and propagate information over longer time intervals, thus mitigating the vanishing gradient problem.

Although such networks can, in principle, capture long-term dependencies, it is known that more abstract and longer-term features are often represented better by deeper architectures (Bengio et al., 2009). To that end, multiple recurrent cells are stacked on top of each other in a feedforward manner, i.e., the output (or the hidden state) of the lower cell is connected to the input gate of the next-higher cell. Many works have used such deep recurrent architectures, e.g., (Chung et al., 2015; Zilly et al., 2017), and have shown their ability to extract more complex features from the input and make better predictions. The need for multi-layer RNNs is particularly apparent for image-like input data, where multiple convolutional layers are required to extract a good representation, while the recurrence captures the evolution of each layer over time.

Since recurrent architectures are trained by propagating gradients across time, it is convenient to "unwrap" them into a lattice with two axes for depth (abstraction level) and time, see Fig. 1. This view makes it apparent that gradients flow in two directions, namely backwards in time and downwards from deeper to shallower layers. In this paper we ask the question *how the basic recurrent unit must be designed to ensure the "vertical" gradient flow across layers is stable* and not impaired

by vanishing or exploding gradients. We show that stacking several layers of common RNN cells, by their construction, leads to instabilities (e.g., for deep LSTMs the gradients tend to vanish; for deep vanilla RNNs they tend to explode). Our study makes three contributions: *(i)* We analyse how the magnitude of the gradient changes as it propagates through a cell of the two-dimensional deep RNN lattice. We show that, depending on the inner architecture of the employed RNN cell, gradients tend to be either amplified or attenuated. As the depth increases, the repeated amplification (resp., attenuation) increases the risk of exploding (resp., vanishing) gradients. *(ii)* We then leverage our analysis to design a new form of gated cell, termed the STAR (stackable recurrent) unit, which better preserves the gradient magnitude inside the RNN lattice. It can therefore be stacked to much greater depth and still remains trainable. *(iii)* Finally, we compare deep recurrent architectures built from different basic cells in an extensive set of experiments with three popular datasets. The results confirm our analysis: training deep recurrent nets fail with most conventional units, whereas the proposed STAR unit allows for significantly deeper architectures. In several cases, the ability to go deeper also leads to improved performance.

2 Related Work

Vanishing or exploding gradients during training are a long-standing problem of recurrent (and other) neural networks (Hochreiter, 1991; Bengio et al., 1994). Perhaps the most effective measure to address them so far has been to introduce gating mechanisms in the RNN structure, as first proposed by (Hochreiter & Schmidhuber, 1997) in the form of the LSTM (long short-term memory), and later by other architectures such as gated recurrent units (GRU, Chung et al., 2014).

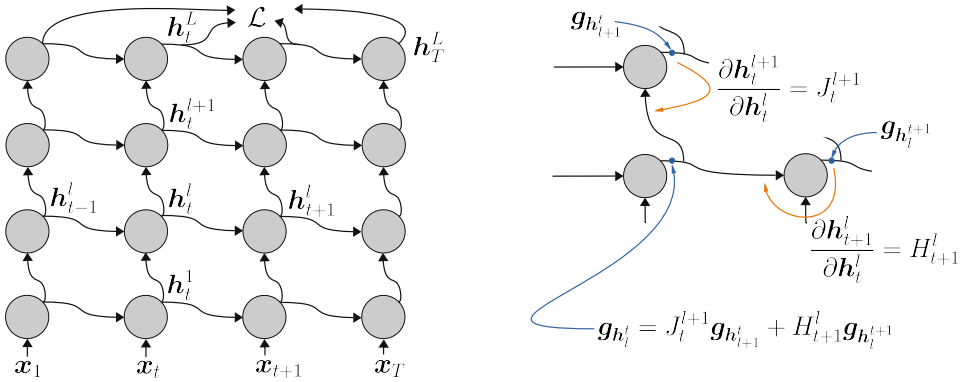
Importantly, RNN training needs proper initialisation. (Le et al., 2015) and (Henaff et al., 2016) have shown that initializing the weight matrices with identity and orthogonal matrices can be useful to stabilise the training. (Arjovsky et al., 2016) and (Wisdom et al., 2016) further develop this idea and impose orthogonality throughout the entire training to keep the amplification factor of the weight matrices close to unity, leading to a more stable gradient flow. Unfortunately, it has been shown (Vorontsov et al., 2017) that such hard orthogonality constraints hurt the representation power of the model and in some cases even *destabilise* the optimisation.

Another line of work has studied ways to mitigate the vanishing gradient problem by introducing additional (skip) connections across time and/or layers. (Campos et al., 2018) have shown that skipping state updates in RNN shrinks the effective computation graph and thereby helps to learn longer-range dependencies. (Kim et al., 2017) introduce a residual connection between LSTM layers. (Chung et al., 2015) propose a gated feedback RNN that extends the stacked RNN architecture with extra connections. An obvious disadvantage of such an architecture are the extra computation and memory costs of the additional connections. Moreover, the authors only report results for rather shallow networks up to 3 layers.

Despite the described efforts, it remains challenging to train deep RNNs. (Zilly et al., 2017) have proposed Recurrent Highway Networks (RHN) that combine LSTMs and highway networks (Srivastava et al., 2015) to train deeper architectures. RHN are popular and perform well on language modelling tasks, but they are still prone to exploding gradients, as illustrated in our experiments. (Li et al., 2018a) propose a restricted RNN where all interactions are removed between neurons in the hidden state of a layer. This appears to greatly reduce the exploding gradient problem (allowing up to 21 layers), at the cost of a much lower representation power per layer.

To process image sequence data, computer vision systems often rely on Convolutional LSTMs (convLSTM, Xingjian et al., 2015). But while very deep CNNs are very effective and now standard (Krizhevsky et al., 2012; Simonyan & Zisserman, 2015), stacks of more than a few convLSTMs do not train well. In practice, shallow versions are preferred, for instance (Li et al., 2018b) use a single layer for action recognition, and (Zhang et al., 2018) use two layers to recognize for hand gestures (combined with a deeper feature extractor without recursion).

We note that attempts to construct a deep counterpart to the Kalman filter can also be interpreted as recurrent networks (Krishnan et al., 2015; Becker et al., 2019; Coskun et al., 2017). These provide a probabilistic, generative perspective on RNNs, but are even more complicated to train. It is at this point unclear how the basic units of these architectures could be stacked into a deep, multi-layer representation.



(a) (b)

Figure 1: (a) General structure of an unfolded deep RNN (b) Detail of the gradient backpropagation in the two dimensional lattice.

3 Background and Problem Statement

A RNN cell is a non-linear transformation that maps the input signal $\boldsymbol{x}_t$ at time t and the hidden state of the previous time step $t-1$ to the current hidden state $\boldsymbol{h}_t$:

$$\boldsymbol{h}_t = f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{W}) \tag{1}$$

with $\boldsymbol{W}$ the trainable parameters of the cell. The input sequences have an overall length of T, which can be variable. It depends on the task whether the relevant target prediction, for which also the loss $\mathcal{L}$ should be computed, is the final state $\boldsymbol{h}_T$, the complete sequence of states $\{\boldsymbol{h}_t\}$, or a single sequence label, typically defined as the average $\frac{1}{T}\sum_t \boldsymbol{h}_t$. Learning amounts to fitting $\boldsymbol{W}$ to minimise the loss, usually with stochastic gradient descent.

When stacking multiple RNN cells on top of each other, one passes the hidden state of the lower level $l-1$ as input to the next-higher level l, see Fig. 1, which in mathematical terms corresponds to the recurrence relation

$$\boldsymbol{h}_t^l = f(\boldsymbol{h}_t^{l-1}, \boldsymbol{h}_{t-1}^l, \boldsymbol{w})\,. \tag{2}$$

Temporal unfolding leads to a two-dimensional lattice with depth L and length T, as in Fig. 1. In this computation diagram, the forward pass runs from left to right and from bottom to top. Gradients flow in opposite direction: at each cell the gradient w.r.t. the loss arrives at the output gate and is used to compute the gradient w.r.t. *(i)* the weights, *(ii)* the input, and *(iii)* the previous hidden state. The latter two gradients are then propagated through the respective gates to the preceding cells in time and depth. In the following, we investigate how the magnitude of these gradients changes across the lattice. The analysis, backed up by numerical simulations, shows that common RNN cells are biased towards attenuating or amplifying the gradients, and thus prone to destabilising the training of deep recurrent networks.

3.1 Gradient Magnitudes

At a single cell in the lattice the gradient w.r.t. the trainable weights are

$$\boldsymbol{g}_{\boldsymbol{w}} = \frac{\partial \boldsymbol{h}_t^l}{\partial \boldsymbol{w}} \boldsymbol{g}_{\boldsymbol{h}_t^l}, \tag{3}$$

where $\frac{\partial \boldsymbol{h}_t^l}{\partial \boldsymbol{w}}$ denotes the Jacobian matrix and $\boldsymbol{g}_{\boldsymbol{h}_t^l}$ is a column vector containing the partial derivatives of the loss w.r.t. the cell's output (hidden) state. From the equation, it becomes clear that the Jacobian acts as a "gain matrix" on the gradients, and should on average preserve their magnitude to prevent them from vanishing or exploding. By expanding the gradient $\boldsymbol{g}_{\boldsymbol{h}_t^l}$ we obtain the recurrence for propagation,

$$\boldsymbol{g}_{\boldsymbol{h}_t^l} = \frac{\partial \boldsymbol{h}_t^{l+1}}{\partial \boldsymbol{h}_t^l} \boldsymbol{g}_{\boldsymbol{h}_t^{l+1}} + \frac{\partial \boldsymbol{h}_{t+1}^l}{\partial \boldsymbol{h}_t^l} \boldsymbol{g}_{\boldsymbol{h}_{t+1}^l} = J_t^{l+1} \boldsymbol{g}_{\boldsymbol{h}_t^{l+1}} + H_{t+1}^l \boldsymbol{g}_{\boldsymbol{h}_{t+1}^l}, \tag{4}$$

with J_t^l the Jacobian w.r.t. the input and H_t^l the Jacobian w.r.t. the hidden state. Ideally we would like the gradient magnitude $\|\boldsymbol{g}_{\boldsymbol{h}_t^l}\|_2$ to remain stable for arbitrary l and t. Characterising that magnitude completely is difficult, since correlations may exist between $\boldsymbol{g}_{\boldsymbol{h}_{t+1}^l}$ and $\boldsymbol{g}_{\boldsymbol{h}_t^{l+1}}$. Still, it is clear that the two Jacobians J_t^{l+1} and H_{t+1}^l play a fundamental role: if their singular values are small, they will attenuate the gradients and cause them to vanish sooner or later. If their singular values are large, they will amplify the gradients and make them explode.[1]

In the following, we analyze the behaviour of the two matrices for two extreme cases. Let us first consider the simplest RNN cell, hereinafter called Vanilla RNN (vRNN). Its recurrence equation reads

$$\boldsymbol{h}_t^l = \tanh(\boldsymbol{W}_x \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_h \boldsymbol{h}_{t-1}^l + \boldsymbol{b}) \tag{5}$$

from which we get the two Jacobians

$$J_t^l = \boldsymbol{D}_{\tanh(\boldsymbol{W}_x \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_h \boldsymbol{h}_{t-1}^l + \boldsymbol{b})'} \boldsymbol{W}_x \tag{6}$$

$$H_t^l = \boldsymbol{D}_{\tanh(\boldsymbol{W}_x \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_h \boldsymbol{h}_{t-1}^l + \boldsymbol{b})'} \boldsymbol{W}_h \tag{7}$$

where $\boldsymbol{D}_{\boldsymbol{x}}$ denotes a diagonal matrix with the elements of vector $\boldsymbol{x}$ as diagonal entries. Ideally, we would like to know the expected values of the two matrices' singular values. Unfortunately, there is no easy way to derive closed-form analytical expressions for them, but we can compute them for a fixed, representative point. Perhaps the most natural and illustrative choice is to set $\boldsymbol{h}_t^{l-1} = \boldsymbol{h}_{t-1}^l = \boldsymbol{b} = 0$, and to further choose orthogonal weight matrices $\boldsymbol{W}_h$ and $\boldsymbol{W}_x$, a popular initialisation strategy. Since the derivative $\tanh'(0) = 1$, the singular values of all matrices in Eq. (7) are equal to 1 in this configuration.

Consequently, by combining the contributions of $\boldsymbol{g}_{\boldsymbol{h}_t^{l+1}}$ and $\boldsymbol{g}_{\boldsymbol{h}_{t+1}^l}$ we expect to obtain a gradient $\boldsymbol{g}_{\boldsymbol{h}_t^l}$ with larger magnitude. A deep network made of vRNN cells and orthogonal initialisation can thus be expected to suffer from exploding gradients as we move towards shallower layers and further back in time. To validate this analysis we set up a toy example of a deep vRNN and compute the average gradient magnitude w.r.t. the network parameters for each cell in the unfolded network. For this numerical simulation we initialise all the hidden states and biases to zero and chose random orthogonal matrices for the weights. Input sequences are generated with the random process $\boldsymbol{x}_t = \alpha \boldsymbol{x}_{t-1} + (1-\alpha)\boldsymbol{z}$, where $\boldsymbol{z} \sim \mathcal{N}(0,1)$ and the correlation factor $\alpha = 0.5$ (the choice of the correlation factor does not seem to affect qualitatively the results). Figure 2 depicts the average gradient magnitudes over 10K runs with different weight initialisations and input sequences. Indeed, the magnitude grows rapidly towards the earlier and shallower part of the network, as expected from our analysis.

We go on to perform a similar analysis for the classical LSTM cell, as proposed by Hochreiter & Schmidhuber (1997, recurrence equations reproduced in the appendix for convenience). The Jacobians in this case are

$$J_t^l = \boldsymbol{D}_{\tanh(\boldsymbol{c}_t^l)} \boldsymbol{D}_{(\boldsymbol{o}_t^l)'} \boldsymbol{W}_{xo} + \boldsymbol{D}_{\tanh(\boldsymbol{c}_t^l)'} \boldsymbol{D}_{\boldsymbol{o}_t^l} (\boldsymbol{D}_{\boldsymbol{c}_{t-1}^l} \boldsymbol{D}_{(\boldsymbol{f}_t^l)'} \boldsymbol{W}_{xf} + \boldsymbol{D}_{\boldsymbol{z}_t^l} \boldsymbol{D}_{(\boldsymbol{i}_t^l)'} \boldsymbol{W}_{xi} + \boldsymbol{D}_{\boldsymbol{i}_t^l} \boldsymbol{D}_{(\boldsymbol{z}_t^l)'} \boldsymbol{W}_{xz}) \tag{8}$$

$$H_t^l = \boldsymbol{D}_{\tanh(\boldsymbol{c}_t^l)} \boldsymbol{D}_{(\boldsymbol{o}_t^l)'} \boldsymbol{W}_{ho} + \boldsymbol{D}_{\tanh(\boldsymbol{c}_t^l)'} \boldsymbol{D}_{\boldsymbol{o}_t^l} (\boldsymbol{D}_{\boldsymbol{c}_{t-1}^l} \boldsymbol{D}_{(\boldsymbol{f}_t^l)'} \boldsymbol{W}_{hf} + \boldsymbol{D}_{\boldsymbol{z}_t^l} \boldsymbol{D}_{(\boldsymbol{i}_t^l)'} \boldsymbol{W}_{xi} + \boldsymbol{D}_{\boldsymbol{i}_t^l} \boldsymbol{D}_{(\boldsymbol{z}_t^l)'} \boldsymbol{W}_{hz}) \tag{9}$$

The equations are slightly more complicated, but are still amenable to the same type of analysis. We again choose the same exemplary conditions as for the vRNN above, i.e., hidden states and biases equal to zero and orthogonal weight matrices. This time we additionally need the value of the sigmoid activation at the considered point, $\sigma(0) = 0.5$. By substituting the numerical values in the above equations we can see that the sigmoid function causes the singular values of the two Jacobians to drop to 0.25. Contrary to the vRNN cell, we expect that even the two Jacobians combined will produce an attenuation factor well below 1, such that the gradient magnitude will rapidly decline and eventually vanish. We point out that t LSTM cells have a second hidden state, the so-called "cell state". The cell state only propagates along the time dimension and not across layers, which makes the overall effect of the corresponding gradients more difficult to analyse. However, for the same reason one would, in a first approximation, expect that the cell state mainly influences the gradients

[1] A subtle point is that sometimes large gradients are the precursor of vanishing gradients if the associated large parameter updates cause the non-linearities to saturate.

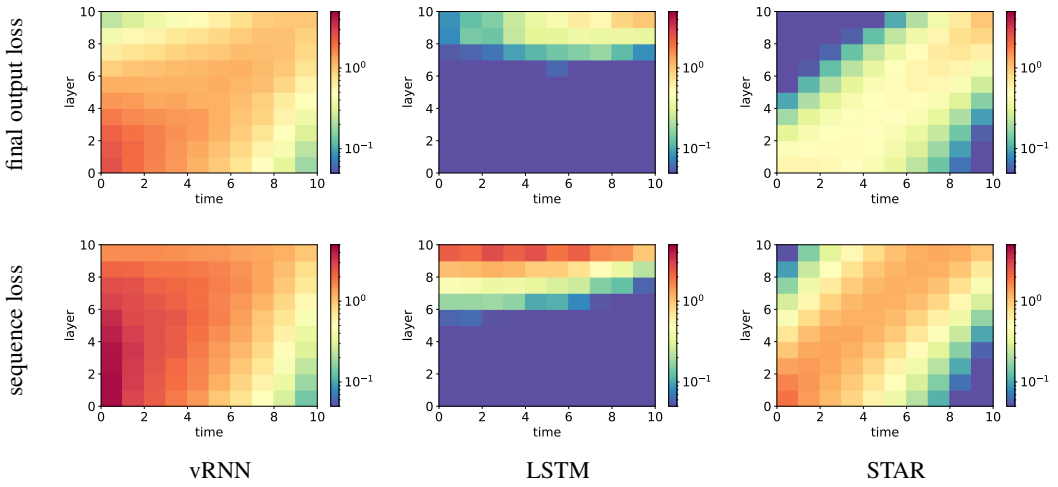


Figure 2: Mean value of gradient magnitude with respect to the parameters for different RNN units. *top row:* loss $\mathcal{L}(h_T^L)$ only on final prediction. *bottom row:* loss $\mathcal{L}(h_1^L \dots h_T^L)$ over all time steps. As the gradients flow back through time and layers, for a network of vanilla RNN units they get amplified; for LSTM units they get attenuated; whereas the proposed STAR unit approximately preserves their magnitude.

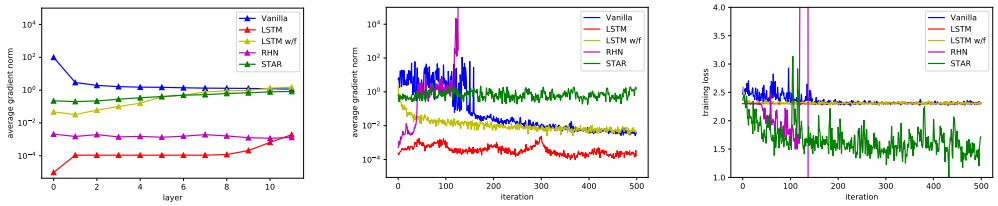
in the time direction, but cannot help the flow through the layers. Again the numerical simulation results support our hypothesis, as can be seen in Fig. 2. The LSTM gradients propagate relatively well backward through time, but vanish quickly towards shallower layers. We refer to the appendix for further numerical analysis, e.g., LSTMs with only a forget gate, and GRUs.

Here, we briefly draw some connections between our analysis and the empirical results of Chung et al. (2015), who propose a gated feedback RNN (GFRNN) that extends the stacked RNN architecture with extra connections between adjacent layers. Empirically, GFRNN improves a 3-layer LSTM, but degrades the vanilla RNN performance. We conjecture that this might be due to the extra connections strengthening the gradient propagation. According to our findings, the additional gradient flow would benefit the LSTM, by bolstering the dwindling gradients; whereas for the vRNN, where the initial gradients are already too high, the added flow might be counterproductive.

4 The STAR Unit

Building on the analysis above, we introduce a novel RNN cell designed to avoid vanishing or exploding gradients as much as possible. We start from the Jacobian matrix of the LSTM cell and examine in more detail which design features are responsible for the low singular values. In equation 8 we see that every multiplication with $\tanh$ non-linearities ($\boldsymbol{D}_{\tanh(.)}$), gating functions ($\boldsymbol{D}_{\sigma(.)}$), and with their derivatives can only ever decrease the singular values of $\boldsymbol{W}$, since all this terms are always <1. The effect is particularly pronounced for the sigmoid and its derivative, $|\sigma'(\cdot)| \leq 0.25$ and $\mathbb{E}[|\sigma'(x)|] = 0.5$ for zero-mean, symmetric distribution of x. In particular, the output gate $\boldsymbol{o}_t^l$ is a sigmoid and plays a major role in shrinking the overall gradients, as it multiplicatively affects all parts of both Jacobians. As a first measure, we thus propose to remove the output gate. A secondary consequence of this measure is that now $\boldsymbol{h}_t^l$ and $\boldsymbol{c}_t^l$ carry the same information (the hidden state becomes an element-wise non-linear transformation of the cell state). To avoid this duplication and further simplify the design, we transfer the $\tanh$ non-linearity to the hidden state and remove the cell state altogether.

As a final modification, we also remove the input gate $\boldsymbol{i}_t^l$ from the architecture. We have empirically observed that the presence of the input gate does not significantly improve performance, moreover, it actually harms the training for deeper networks. This empirical observation is in line with the results of van der Westhuizen & Lasenby (2018), who show that removing the input and output gates does not greatly affect the performance of LSTMs.



(a) gradient norm / layer (b) gradient norm, 1st epoch (c) training loss, 1st epoch

Figure 3: Gradient magnitudes in pix-by-pix MNIST. (a) Mean gradient norm per layer at the start of training. (b) Evolution of gradient norm during 1st training epoch. (c) Loss during 1st epoch.

More formally, our proposed STAR cell in the l-th layer takes the input $\boldsymbol{h}_t^{l-1}$ (in the first layer, $\boldsymbol{x}_t$) at time t and non-linearly projects it to the space where the hidden vector $\boldsymbol{h}^l$ lives, equation 10. Furthermore, the previous hidden state and the new input are combined into the gating variable $\boldsymbol{k}_t^l$ (equation 11). $\boldsymbol{k}_t^l$ is our analogue of the forget gate and controls how the information from previous hidden state and the new input are fused into a new hidden state. One could also intuitively interpret $\boldsymbol{k}_t^l$ as a sort of "Kalman gain": if it is large, the new observation is deemed reliable and dominates; otherwise the previous hidden state is conserved. The complete dynamics of the STAR unit is given by the expressions

$$\boldsymbol{z}_t^l = \tanh(\boldsymbol{W}_z \boldsymbol{h}_t^{l-1} + \boldsymbol{b}_z) \tag{10}$$

$$\boldsymbol{k}_t^l = \sigma(\boldsymbol{W}_x \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_h \boldsymbol{h}_{t-1}^l + \boldsymbol{b}_k) \tag{11}$$

$$\boldsymbol{h}_t^l = \tanh\left((1 - \boldsymbol{k}_t^l) \circ \boldsymbol{h}_{t-1}^l + \boldsymbol{k}_t^l \circ \boldsymbol{z}_t^l\right). \tag{12}$$

These equations lead to the following Jacobian matrices:

$$J_t^l = \boldsymbol{D}_{\tanh(\boldsymbol{h}_{t-1}^l + \boldsymbol{k}_t^l \circ (\boldsymbol{z}_t^l - \boldsymbol{h}_{t-1}^l))'} (\boldsymbol{D}_{\boldsymbol{z}_t^l - \boldsymbol{h}_{t-1}^l} \boldsymbol{D}_{(\boldsymbol{k}_t^l)'} \boldsymbol{W}_x + \boldsymbol{D}_{\boldsymbol{k}_t^l} \boldsymbol{D}_{(\boldsymbol{z}_t^l)'} \boldsymbol{W}_z) \tag{13}$$

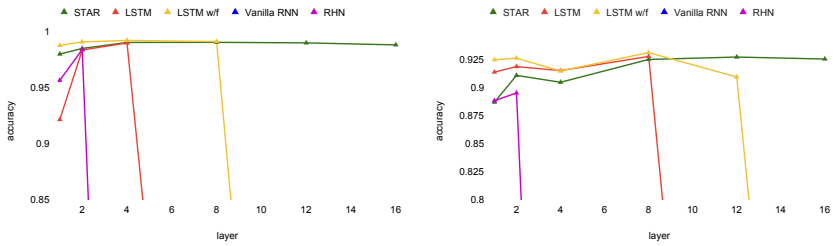
$$H_t^l = \boldsymbol{D}_{\tanh(\boldsymbol{h}_{t-1}^l + \boldsymbol{k}_t^l \circ (\boldsymbol{z}_t^l - \boldsymbol{h}_{t-1}^l))'} (\boldsymbol{I} + \boldsymbol{D}_{\boldsymbol{z}_t^l - \boldsymbol{h}_{t-1}^l} \boldsymbol{D}_{(\boldsymbol{k}_t^l)'} \boldsymbol{W}_h - \boldsymbol{D}_{\boldsymbol{k}_t^l}). \tag{14}$$

Coming back to our previous analysis for state zero and orthogonal weight matrices, each of the two Jacobians now has singular values equal to 0.5. I.e., they lie between the vRNN cell and the LSTM cell, and when added together roughly preserve the gradient magnitude. We repeat the same numerical simulations as above for the STAR cell, and find that it indeed maintains healthy gradient magnitudes throughout most of the deep RNN, see Fig. 2. In the next section, we show also on real datasets that deep RNNs built from STAR units can be trained to a significantly greater depth.

As a final remark, the proposed modifications mean that the STAR architecture requires significantly less memory. With the same input and the same capacity in the hidden state, it reduces the memory footprint to <40% of a classical LSTM and even uses slightly less memory than a recurrent highway net. A more detailed comparison is given in the appendix.

5 Experiments

We evaluate the performance of several well-known RNN baselines as well as that of the proposed STAR cell on five different sequence modelling tasks with three different datasets: sequential versions of *MNIST*, which are a popular common testbed for recurrent networks; the more realistic *TUM dataset*, where time series of intensities observed in satellite images shall be classified into different agricultural crops; and Jester, for hand gesture recognition with convolutional RNNs. The recurrent units we compare include the vRNN, the LSTM, the LSTM with only a forget gate (van der Westhuizen & Lasenby, 2018), the RHN, and the proposed STAR. The experimental protocol is similar for all tasks: For each RNN variant, we train multiple versions with different depth (number of layers). For each variant and each depth, we report the performance of the model with the lowest validation loss. Classification performance is measured by the rate of correct predictions (top-1 accuracy). Throughout, we use the orthogonal initialisation for weight matrices. Code and trained models (in Tensorflow), as well as code for the simulations (in PyTorch), will be released. Training and network details for each experiment can be found in the appendix.



(a) MNIST (b) pMNIST

Figure 4: Results for pixel-by-pixel MNIST tasks.

Table 1: Performance comparison for pixel-by-pixel MNIST tasks.

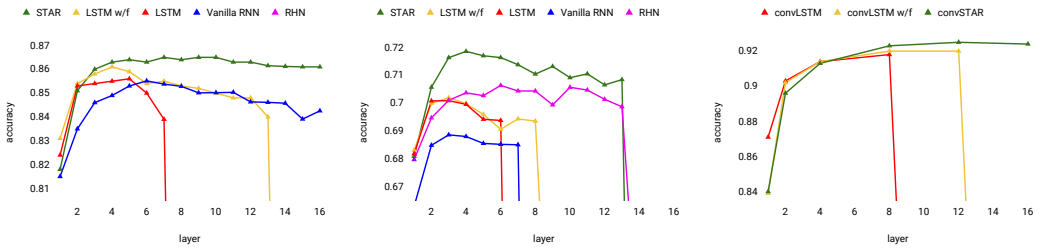
Method	MNIST	pMNIST	#units	#params
vRNN (1 layers)	24.3%	44.0%	128	18k
LSTM (4 layers)	99.0%	91.5%	128	463k
iRNN (Le et al., 2015)	97.0%	82.0%	100	11k
uRNN (Arjovsky et al., 2016)	95.1%	91.4%	512	9k
FC uRNN (Wisdom et al., 2016)	96.9%	94.1%	512	270k
Soft ortho (Vorontsov et al., 2017)	94.1%	91.4%	128	18k
AntisymRNN (Chang et al., 2019)	98.8%	93.1%	128	10k
STAR (4 layers)	99.1%	90.5%	128	166k
STAR (16 layers)	99.0%	92.4%	64	190k

5.1 PIXEL-BY-PIXEL MNIST

The first experiment uses the MNIST dataset (LeCun et al., 1998). The 28×28 grey-scale images of handwritten digits are flattened into 784×1 vectors, and the 784 values are sequentially presented to the RNN. After seeing all pixels, the model predicts the digit. The second task, pMNIST, is more challenging. Before flattening the images, the pixels are shuffled with a fixed random permutation, turning correlations between spatially close pixels into non-local long-range dependencies. The model needs to remember those dependencies between distance parts of the sequence to classify the digit correctly. Fig. 3a shows the average gradient norms per layer at the start of training, for 12-layer networks built from different RNN cells. Like in the simulations above, the propagation through the network increases the gradients for the vRNN and shrinks them for the LSTM. As the optimisation proceeds, we find that STAR remains stable, whereas all other units see a rapid decline of the gradients already within the first epoch, except for RHN, where the gradients explode, see Fig. 3b. Consequently, STAR is the only unit for which a 12-layer model can be trained, as also confirmed by the evolution of the training loss, Fig. 3c. Fig. 4 confirms that stacking into deeper architectures does benefit RNNs (except for vRNN); but it increases the risk of a catastrophic training failure. STAR is significantly more robust in that respect and can be trained up to a depth of 20 layers. On the comparatively easy and saturated MNIST data, the performance is comparable that of a successfully trained LSTM (at depth 2-8 layers, LSTM training already often fails; the displayed accuracies are averaged only over successful training runs).

5.2 TUM TIME SERIES CLASSIFICATION

In this experiment, the models are evaluated on a more realistic sequence modelling problem. The task is to classify agricultural crop types using sequences of satellite images, exploiting the fact that different crops have different growing patterns over the season. The input is a time series of 26 multi-spectral Sentinel-2A satellite images with a ground resolution of 10 m, collected over a 102 km x 42 km area north of Munich, Germany between December 2015 and August 2016 (Rußwurm & Körner, 2017). The input data points for the classifier are patches of 3×3 pixels recorded in



(a) TUM, per time-step labels (b) TUM, single label / sequence (c) Jester, single label / sequence

Figure 5: Time series classification. *(a,b)* Crop classes. *(c)* Hand gestures (convolutional RNNs).

6 spectral channels, flattened into 54×1 vectors. In the first task these vectors are sequentially presented to the RNN model, which outputs a prediction at every time step (note that for this task the correct answer can sometimes be "cloud", "snow", "cloud shadow" or "water", which are easier to recognise than many crops). In the second task, the model makes only one crop prediction for the complete sequence, via an additional layer that averages across time. From Fig. 5 we see that STAR outperforms all baselines and its again more robust to stacking. For the single-output task also the STAR network fails at 14 layers. We have not yet been able to identify the reason for this, possibly it is due to cloud cover that completely blanks out the signal over extended time windows and degrades the propagation.

5.3 Hand-gesture recognition from video

This experiment serves to evaluate the performance of different recurrent cells, and in particular the proposed STAR cell, in a convolutional RNN (see appendix for details about convolutional STAR). To that end, we use the 20BN-Jester dataset V1 (jes). Jester is a large collection of densely-labeled short video clips, where each clip contains a predefined hand gesture performed by a worker in front of a laptop camera or webcam. In total, the dataset includes 148'094 RGB video files of 27 types of gestures. The task is to classify which gesture is seen in a video. 32 consecutive frames of size 112×112 pixels are sequentially presented to the convolutional RNN. At the end, the model again predicts a gesture class via an averaging layer over all time steps. The outcome for convolutional RNNs is coherent with the previous results, see Fig. 5c. Going deeper improves the performance of all three tested convRNNs. The improvement is strongest for convolutional STAR, and the best performance is reached at high depth (12 layers), where training the baselines mostly fails. In summary, the results confirm both our intuition that depth is particularly useful for convolutional RNNs; and that STAR is more suitable for deeper architectures, where it achieves higher performance with better memory efficiency. We note that the in the shallow 1-2 layer setting the conventional LSTM performs a bit better than the two others, likely due to its larger capacity.

6 Conclusion

We have investigated the problem of vanishing/exploding gradient in deep RNNs. In a first step, we analyse how the derivatives of the non-linear activation functions rescale the gradients as they propagate through the temporally unrolled network. From both, the theoretical analysis, and associated numerical simulations, we find that standard RNN cells do not preserve the gradient magnitudes during backpropagation, and therefore, as the depth of the network grows, the risk that the gradients vanish or explode increases. In a second step, we have proposed a new RNN cell, termed the STAckable Recurrent unit, which better preserves gradients through deep architectures and facilitates their training. An extensive evaluation on three popular datasets confirms that STAR units can be stacked into deeper architectures than other RNN cells.

We see two main directions for future work. On the one hand, it would be worthwhile to develop a more formal and thorough mathematical analysis of the gradient flow, and perhaps even derive rigorous bounds for specific cell types, that could, in turn, inform the network design. On the other hand, it appears promising to investigate whether the analysis of the gradient flows could serve as a basis for better initialisation schemes to compensate the systematic influences of the cells structure, e.g., gating functions, in the training of deep RNNs.

REFERENCES

The 20bn-jester dataset v1. https://20bn.com/datasets/jester.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *ICML*, 2016.

Philipp Becker, Harit Pandya, Gregor Gebhardt, Cheng Zhao, C James Taylor, and Gerhard Neumann. Recurrent kalman networks: Factorized inference in high-dimensional deep feature spaces. In *ICML*, 2019.

Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE TNN*, 5(2):157–166, 1994.

Yoshua Bengio et al. Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

Víctor Campos, Brendan Jou, Xavier Giró-i Nieto, Jordi Torres, and Shih-Fu Chang. Skip RNN: Learning to skip state updates in recurrent neural networks. In *ICLR*, 2018.

Bo Chang, Minmin Chen, Eldad Haber, and Ed H Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *ICLR*, 2019.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *ICML*, 2015.

Huseyin Coskun, Felix Achilles, Robert DiPietro, Nassir Navab, and Federico Tombari. Long short-term memory Kalman filters: Recurrent neural estimators for pose regularization. In *ICCV*, 2017.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. *arXiv preprint arXiv:1602.06662*, 2016.

Sepp Hochreiter. Untersuchungen zu Dynamischen Neuronalen Netzen. *Diploma Thesis, Technische Universität München*, 91(1), 1991.

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

Jaeyoung Kim, Mostafa El-Khamy, and Jungwon Lee. Residual LSTM: Design of a deep recurrent architecture for distant speech recognition. *arXiv preprint arXiv:1701.03360*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.

Rahul G. Krishnan, Uri Shalit, and David Sontag. Deep Kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *CVPR*, 2018a.

Zhenyang Li, Kirill Gavrilyuk, Efstratios Gavves, Mihir Jain, and Cees GM Snoek. VideoLSTM convolves, attends and flows for action recognition. *CVIU*, 166:41–50, 2018b.

Marc Rußwurm and Marco Körner. Temporal vegetation modelling using long short-term memory networks for crop identification from medium-resolution multi-spectral satellite images. In *CVPR Workshops*, 2017.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Corentin Tallec and Yann Ollivier. Can recurrent neural networks warp time? In *ICLR*, 2018.

Jos van der Westhuizen and Joan Lasenby. The unreasonable effectiveness of the forget gate. *CoRR*, abs/1804.04849, 2018.

Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. On orthogonality and learning recurrent networks with long term dependencies. In *ICML*, 2017.

Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *NIPS*, 2016.

Shi Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *NIPS*, 2015.

Liang Zhang, Guangming Zhu, Lin Mei, Peiyi Shen, Syed Afaq Ali Shah, and Mohammed Bennamoun. Attention in convolutional LSTM for gesture recognition. In *NIPS*, 2018.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. In *ICML*, 2017.

A RNN CELLS DYNAMICS

A.1 VANILLA RNN

Vanilla RNN update rule:

$$\boldsymbol{h}_t^l = \tanh(\boldsymbol{W}_x \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_h \boldsymbol{h}_{t-1}^l + \boldsymbol{b}) \tag{15}$$

A.2 LSTM

LSTM update rule:

$$\boldsymbol{i}_t^l = \sigma(\boldsymbol{W}_{xi} \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_{hi} \boldsymbol{h}_{t-1}^l + \boldsymbol{b}_i) \tag{16}$$
$$\boldsymbol{f}_t^l = \sigma(\boldsymbol{W}_{xf} \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_{hf} \boldsymbol{h}_{t-1}^l + \boldsymbol{b}_f) \tag{17}$$
$$\boldsymbol{o}_t^l = \sigma(\boldsymbol{W}_{xo} \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_{ho} \boldsymbol{h}_{t-1}^l + \boldsymbol{b}_o) \tag{18}$$
$$\boldsymbol{z}_t^l = \tanh(\boldsymbol{W}_{xz} \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_{hz} \boldsymbol{h}_{t-1}^l + \boldsymbol{b}_z) \tag{19}$$
$$\boldsymbol{c}_t^l = \boldsymbol{f}_t^l \circ \boldsymbol{c}_{t-1}^l + \boldsymbol{i}_t^l \circ \boldsymbol{z}_t^l \tag{20}$$
$$\boldsymbol{h}_t^l = \boldsymbol{o}_t^l \circ \tanh(\boldsymbol{c}_t^l). \tag{21}$$

A.3 LSTM WITH ONLY FORGET GATE

LSTM with only forget gate update rule:

$$\boldsymbol{f}_t^l = \sigma(\boldsymbol{W}_{xf} \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_{hf} \boldsymbol{h}_{t-1}^l + \boldsymbol{b}_f) \tag{22}$$
$$\boldsymbol{z}_t^l = \tanh(\boldsymbol{W}_{xz} \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_{hz} \boldsymbol{h}_{t-1}^l + \boldsymbol{b}_z) \tag{23}$$
$$\boldsymbol{h}_t^l = \tanh(\boldsymbol{f}_t^l \circ \boldsymbol{h}_{t-1}^l + (1 - \boldsymbol{f}_t^l) \circ \boldsymbol{z}_t^l) \tag{24}$$

A.4 GRU

GRU update rule:

$$\boldsymbol{z}_t^l = \sigma(\boldsymbol{W}_{xz} \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_{hz} \boldsymbol{h}_{t-1}^l + \boldsymbol{b}_z) \tag{25}$$
$$\boldsymbol{r}_t^l = \sigma(\boldsymbol{W}_{xr} \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_{hr} \boldsymbol{h}_{t-1}^l + \boldsymbol{b}_r) \tag{26}$$
$$\boldsymbol{h}_t^l = (1 - \boldsymbol{z}_t^l) \circ \boldsymbol{h}_{t-1}^l + \boldsymbol{z}_t^l \circ \tanh\left(\boldsymbol{W}_{xh} \boldsymbol{h}_t^{l-1} + \boldsymbol{W}_{hh}(\boldsymbol{r}_t^l \circ \boldsymbol{h}_{t-1}^l) + \boldsymbol{b}_h\right) \tag{27}$$

A.5 CONVOLUTIONAL STAR

We briefly describe the convolutional version of our proposed cell. The main difference is matrix multiplications now become convolution operation. The dynamics of the convSTAR cell is given in the following equations.

$$\mathbf{K}_t^l = \sigma(\mathbf{W}_x * \mathbf{H}_t^{l-1} + \mathbf{W}_h * \mathbf{H}_{t-1}^l + \mathbf{B}_K) \tag{28}$$
$$\mathbf{Z}_t^l = \tanh(\mathbf{W}_z * \mathbf{H}_t^{l-1} + \mathbf{B}_z) \tag{29}$$
$$\mathbf{H}_t^l = \tanh(\mathbf{H}_{t-1}^l + \mathbf{K}_t^l \circ (\mathbf{Z}_t^l - \mathbf{H}_{t-1}^l)) \tag{30}$$

B FURTHER NUMERICAL GRADIENT PROPAGATION ANALYSIS

In this section we extend the numerical simulations of the gradient propagation in the unfolded recurrent neural network to two further cell architectures, namely the GRU (Chung et al., 2014) and the LSTM with only forget gate (dynamics can be found in Appendix A). The setup of the numerical simulations is the same as the one described in Section 3. As can be seen from Fig. 6 the GRU and the LSTM with only forget gate mitigate the attenuation of gradients to some degree. However, we observe that the corresponding standard deviations are much higher, i.e., the gradient norm greatly varies across different runs, see Fig. 7. We found that the gradients within a single run oscillate a lot more, for both LSTMw/f and GRU, and make training unstable which is undesirable. Moreover, the gradient magnitudes evolve very differently for different initial values, meaning that the training is less robust against fluctuations of the random initialisation.

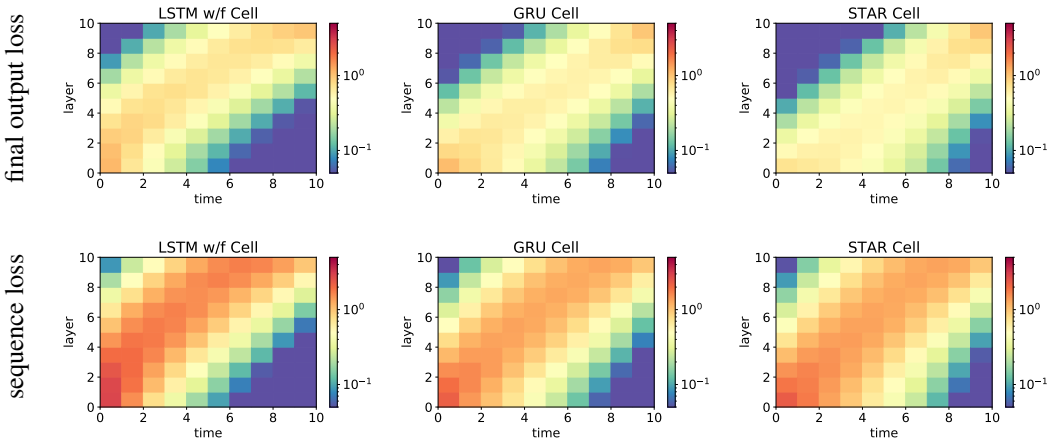


Figure 6: Mean gradient magnitude w.r.t. the parameters for LSTM with only forget gate, GRU, and the proposed STAR cell. *top row:* loss $\mathcal{L}(h_T^L)$ only on final prediction. *bottom row:* loss $\mathcal{L}(h_1^L \dots h_T^L)$ over all time steps.

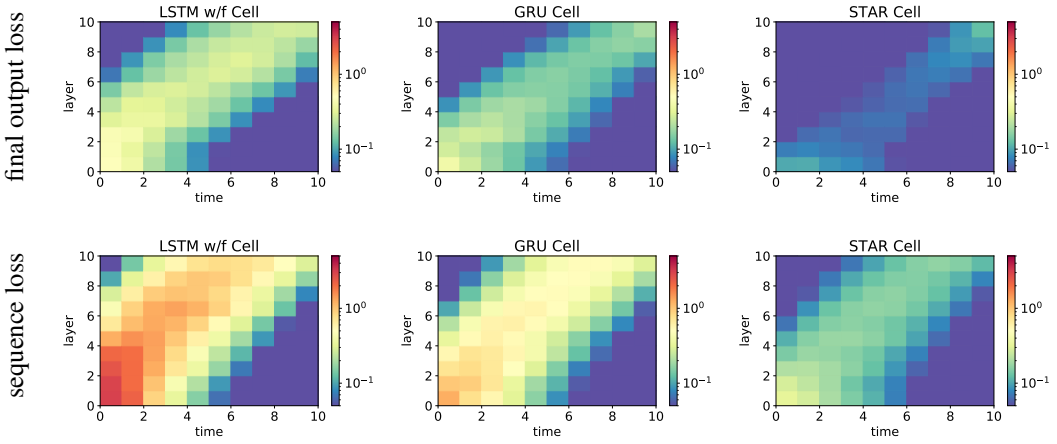


Figure 7: Mean-normalised standard deviation of gradient magnitude for LSTM with only forget gate, GRU, and the proposed STAR cell. *top row:* loss $\mathcal{L}(h_T^L)$ only on final prediction. *bottom row:* loss $\mathcal{L}(h_1^L \dots h_T^L)$ over all time steps.

C Training details

C.1 Pixel-by-pixel MNIST

Following Tallec & Ollivier, chrono initialisation is applied for the bias term of $\boldsymbol{k}$, $\boldsymbol{b}_k$. The basic idea is that $\boldsymbol{k}$ should not be too large; such that the memory $\boldsymbol{h}$ can be retained over longer time intervals. The same initialisation is used for the input and forget bias of the LSTM and the RHN and for the forget bias of LSTMw/f. For the final prediction, a feedforward layer with softmax activation converts the hidden state to a class label. The numbers of hidden units in the RNN layers are set to 128. All networks are trained for 100 epochs with batch size 100, using the Adam optimizer (Kingma & Ba, 2014) with learning rate 0.001, $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

C.2 TUM time series classification

For both tasks we use the same training schedule. Again a feedforward layer is appended to the RNN output to obtain a prediction. The numbers of hidden units in the RNN layers is set to 128.

All networks are trained for 30 epochs with batch size 500, using Adam (Kingma & Ba, 2014) with learning rate 0.001 and $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

C.3 HAND-GESTURE RECOGNITION FROM VIDEO

Throughout, convolution kernels are of size 3×3. Each convolutional RNN layer has 64 filters. A shallow CNN is used to convert the hidden state to a label, with 4 layers that have filter depths 128, 128, 256 and 256, respectively. All models are fitted with stochastic gradient descent (SGD) with momentum ($\beta = 0.9$). The batch size is set to 8, the learning rate starts at 0.001 and decays polynomially to 0.000001 over a total of 30 epochs. $L2$-regularisation with weight 0.00005 is applied to all parameters.

D MEMORY & COMPUTE COMPARISON

Table 2: Parameter counts ($\times10^3$) and floating point operations (#mult&add, $\times10^6$) for pixel-by-pixel MNIST.

#layers	vRNN		LSTM		LSTM w/f		RHN		*STAR*	
1	18	0.30	68	1.11	35	0.58	51	0.86	18	0.30
2	51	0.85	199	3.37	100	1.70	101	1.70	67	1.21
4	116	1.97	463	7.84	232	3.93	200	3.37	166	3.01
8	248	4.20	989	16.77	495	8.39	398	6.72	364	6.62
16	514	8.66	2042	34.62	1021	17.32	795	13.85	759	13.43

E EXPERIMENTAL RESULTS

Table 3: Results for pixel-by-pixel MNIST

	MNIST					**pMNIST**				
#layers	vRNN	LSTM	LSTM w/f	RHN	*STAR*	vRNN	LSTM	LSTM w/f	RHN	*STAR*
1	**43.1**	92.2	98.8	95.5	98.0	**77.3**	91.4	92.5	88.8	88.7
2	10.1	98.4	99.1	**98.4**	98.5	72.1	91.8	92.6	**89.5**	91.1
4	11.3	**99.0**	**99.2**	0	**99.1**	18.5	91.5	91.5	0	90.5
8	11.0	19.3	99.1	0	99.1	9.7	**92.8**	**93.1**	0	92.5
12	11.0	11.0	9.9	0	99.0	9.8	11.3	90.9	0	**92.8**
16	11.1	11.0	11.3	0	99.0	9.7	11.3	11.3	0	92.6

Table 4: Results for TUM time series classification

	per-timestep prediction					**per-sequence prediction**				
#layers	vRNN	LSTM	LSTM w/f	RHN	*STAR*	vRNN	LSTM	LSTM w/f	RHN	*STAR*
1	81.5	82.4	83.1	**72.5**	81.8	66.3	68.2	68.3	68.0	68.1
2	83.5	85.3	85.4	59.5	85.1	68.5	**70.1**	**70.0**	69.5	70.6
4	84.9	**85.5**	**86.1**	70.3	86.3	**68.8**	70.0	70.0	70.3	**71.9**
8	**85.2**	47.2	85.3	67.8	**86.5**	33.8	33.8	69.4	**70.4**	71.0
12	84.6	47.2	84.8	69.0	86.3	33.7	33.8	34.9	70.1	70.7
16	84.2	42.5	47.5	68.6	86.1	33.7	33.8	34.8	50.6	38.7

Table 5: Results for hand-gesture recognition

#layers	convLSTM	convLSTM w/f	*convSTAR*
1	87.1	83.9	84.0
2	90.3	90.2	89.6
4	91.4	91.4	91.3
8	**91.8**	**92.0**	92.3
12	9.0	92.0	**92.5**
16	9.0	9.9	92.4