Gating Revisited: Deep Multi-layer RNNs That Can Be Trained

Mehmet Ozgur Turkoglu, Stefano D’Aronco, Jan Dirk Wegner, Konrad Schindler
EcoVision Lab, ETH Zurich

Abstract

We propose a new stackable recurrent cell (STAR) for recurrent neural networks (RNNs) that has significantly less parameters than widely used LSTM [14] and GRU [8] while being more robust against vanishing or exploding gradients. Stacking multiple layers of recurrent units has two major drawbacks: i) many recurrent cells (e.g., LSTM cells) are extremely eager in terms of parameters and computation resources, ii) deep RNNs are prone to 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 in the "vertical" direction. We show that, depending on the structure of the basic recurrent unit, the gradients are systematically attenuated or amplified. Based on our analysis we design a new type of gated cell that better preserves gradient magnitude. We validate our design on a large number of sequence modelling tasks and demonstrate that the proposed STAR cell allows to build and train deeper recurrent architectures, ultimately leading to improved performance while being computationally efficient.

1. Introduction

Recurrent Neural Networks (RNN) have established themselves as a powerful tool for modelling sequential data. They have led to significant progress for a variety of applications, notably language processing and speech recognition [29, 11, 32].

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 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 [14] and Gated Recurrent Units (GRU) [8]. They use gating mechanisms to store and propagate information over longer time intervals, thus mitigating the vanishing gradient problem.

In general, abstract features are often represented better by deeper architectures [3]. In the same way that multiple hidden layers can be stacked in traditional feed-forward networks, multiple recurrent cells can also be stacked on top of each other, i.e., the output (or the hidden state) of the lower cell is connected to the input gate of the next-higher cell. Several works have shown the ability of deeper recurrent architectures to extract more complex features from the input to make better predictions like [9, 38, 26]. However, such architectures are usually composed of just two or three layers because training deeper recurrent architectures still presents an open problem. More specifically, deep RNNs suffer from two main shortcomings: i) they are difficult to train because of gradient instability, i.e., the gradient either explodes or vanishes during training; and ii) the large number of parameters contained in each single cell makes deep architectures extremely resource-intensive. Both issues restrict the practical use of deep RNNs and particularly their usage for image-like input data, which generally requires multiple convolutional layers to extract discriminative, abstract representations. Our work aims to address these weaknesses by designing a recurrent cell that, on the one hand, requires fewer parameters and, on the other hand, allows for stable gradient back-propagation during training thus allowing for deeper architectures.

Contributions (i) We present a detailed, theoretical analysis of how the gradient magnitude changes as it propagates through a cell in a deep RNN lattice. Our analysis offers a different perspective compared to existing literature about RNN gradients, as it focuses on the gradient flow across layers in depth direction, rather than the recurrent flow across time. We show that the two dimensions behave differently, i.e., the ability to preserve gradients in time direction does not necessarily mean that they are preserved across the layers, too. (ii) We leverage our analysis to design a new, lightweight gated cell, termed the STAckale Recurrent (STAR) unit. The STAR cell better preserves the
2. Related Work

Vanishing or exploding gradients during training are a long-standing problem of recurrent (and other) neural networks [13, 4]. Perhaps the most effective measure to address them so far has been to introduce gating mechanisms in the RNN structure, as first proposed by [14] in the form of the LSTM (long short-term memory), and later by other architectures such as gated recurrent units [8].

Importantly, RNN training needs proper initialisation. [18] and [12] have shown that initialising the weight matrices with identity and orthogonal matrices can be useful to stabilise the training. [2] and [34] 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 [33] 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. [5] have shown that skipping state updates in RNN shrinks the effective computation graph and thereby helps to learn longer-range dependencies. [15, 23] introduce a residual connection between LSTM layers; however, the performance improvements are limited. [9] 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.

Many of the aforementioned works propose new RNN architectures by leveraging a gradient propagation analysis. However, all of these studies, as well as other studies which specifically aim at modelling accurately gradient propagation in RNNs, see [2, 22, 7], overlook the propagation of the gradient in the vertical dimension. In this work we will employ similar gradient analysis techniques but focusing on the depth dimension of the network.

Despite the described efforts, it remains challenging to train deep RNNs. [38] have proposed Recurrent Highway Networks (RHN) that combine LSTMs and highway networks [28] 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. [20] 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, at the cost of a much lower representation power per layer.

To process image sequence data, computer vision systems often rely on Convolutional LSTMs [35]. But while very deep CNNs are very effective and now standard [17, 27], stacks of more than a few convLSTMs do not train well and, because of the large numbers of parameters in a LSTM cell, it also increases drastically the computational demand. In practice, shallow versions are preferred like [21] who use a single layer for action recognition, and [37] who use two layers to recognise hand gestures (combined with a deeper feature extractor without recursion).

3. Background and Problem Statement

In this section we revisit the mathematics of RNNs with particular emphasis on the gradient propagation. We will then leverage this analysis to design a more stable recurrent cell, which is described in Sec. 4. A RNN cell is a non-linear transformation that maps the input signal $x_t$ at time $t$ and the hidden state of the previous time step $t - 1$ to the current hidden state $h_t$:

$$h_t = f(x_t, h_{t-1}, W)$$

with $W$ the trainable parameters of the cell. The input sequences have an overall length of $T$, which can vary. It depends on the task whether the final state $h_T$, the complete sequence of states $\{h_t\}$, or a single sequence label (typically defined as the average $\frac{1}{T} \sum_t h_t$) are the desired target prediction for which loss $L$ is computed. Learning amounts to fitting $W$ to minimise the loss, usually with stochastic gradient descent.

When stacking multiple RNN cells on top of each other, the hidden state of the lower layer $l - 1$ is passed on as input to the next-higher level $l$ (Fig. 1). In mathematical terms this corresponds to the recurrence relation

$$h^l_i = f(h^{l-1}_{i-1}, h^{l-1}_i, w).$$

Temporal unfolding leads to a two-dimensional lattice with depth $L$ and length $T$ (Fig. 1), 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 anal-
to prevent them from vanishing or exploding. We obtain the gradients, and should on average preserve their magnitude. It becomes apparent that the Jacobian acts as a "gain matrix" on the cell’s output (hidden) state. From the equation, it becomes clear that initialisation can thus be expected to suffer, especially in the initial training phase, from exploding gradients as we move towards shallower layers and further back in time. If their singular values are large, 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.

![Image](https://via.placeholder.com/150)

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

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

The gradient w.r.t. the trainable weights at a single cell in the lattice is

\[
 g_w = \frac{\partial h_t^l}{\partial w} g_{h_t^l}, \tag{3}
\]

where \( \frac{\partial h_t^l}{\partial w} \) denotes the Jacobian matrix and \( g_{h_t^l} \) 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 initialisation can thus be expected to suffer, especially in the initial training phase, from exploding gradients as we move towards shallower layers and further back in time. To validate this assumption, 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. We initialise all the hidden states and biases to zero and chose random orthogonal matrices for the weights.

To validate this assumption, 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. We initialise all the hidden states and biases to zero and chose random orthogonal matrices for the weights.

\[
 J_t^l = D_{\text{tanh}(W_x h_t^l-1 + W_h h_{t-1}^l + b)} W_x \tag{6}
\]

\[
 H_t^l = D_{\text{tanh}(W_x h_t^l-1 + W_h h_{t-1}^l + b)} W_h \tag{7}
\]

where \( D_{x} \) denotes a diagonal matrix with the elements of vector \( x \) 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 a 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 \( h_t^l-1 = h_{t-1}^l = b = 0 \), and to further choose weight matrices \( W_x \) and \( W_h \) with average singular value equal to one (different popular initialisation strategies, such as orthogonal and identity matrices, are aligned with this assumption). Since the derivative \( \text{tanh}'(0) = 1 \), the average singular values of all matrices in Eq. (7) are equal to 1 in this configuration.

We expect to obtain a gradient \( g_{h_t^l} \) with a larger magnitude by combining the contributions of \( g_{h_t^l+1} \) and \( g_{h_t^l-1} \). A deep network made of vRNN cells with orthogonal or identity initialisation can thus be expected to suffer, especially in the initial training phase, from exploding gradients as we move towards shallower layers and further back in time. To validate this assumption, 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. We initialise all the hidden states and biases to zero and chose random orthogonal matrices for the weights.

\[ A \text{ subtle point is that sometimes large gradients are the precursor of vanishing gradients if the associated large parameter updates cause the non-linearities to saturate.} \]
for this numerical simulation. Input sequences are generated with random process \( x_t = \alpha x_{t-1} + (1 - \alpha) z \), where \( z \sim \mathcal{N}(0, 1) \) and the correlation factor \( \alpha = 0.5 \) (the choice of the correlation factor does not seem to qualitatively affect the results). Figure 2 depicts average gradient magnitudes over 10K runs with different weight initialisations and input sequences. As expected, the magnitude grows rapidly towards the earlier and shallower part of the network.

We perform a similar analysis for the classical LSTM cell [14]. The expressions for the Jacobians in this case are reported in Eq. (8)-(9). 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. By substituting the numerical values in the aforementioned equations, we can see that the sigmoid function causes the expected singular value 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 decline and eventually vanish. We point out that 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 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.

In summary, the gradient propagation in time and depth directions are two different matters. When considering the latter we need to take into consideration the gradient of the output with respect to the input state, too, and not exclusively consider the gradient w.r.t. the previous hidden state. Moreover, we need to take into account that the output of each cell is connected to two cells rather than one adjacent cell. Note that this analysis is valid if the loss is computed only using the final state \( T \) or if all of them are used (Fig. 2). In the latter case, we simply need to sum the contribution of all the separate losses. Usually, parameters are shared among different instants \( t \) in RNNs but mostly not among different layers. If parameters are shared among different time steps gradients accumulate row-wise (Fig. 2) increasing the magnitude of the gradient w.r.t. to the parameters. This, however, is not true in the vertical direction as weights are not shared. As a consequence, it is particularly important to ensure that the gradient magnitude is preserved between adjacent layers.

4. The STAR Unit

Building upon our previous analysis, we introduce a novel RNN cell designed to avoid vanishing or exploding gradients while reducing the number of parameters. We start from the Jacobian matrix of the LSTM cell and investigate what design features are responsible for the low singular values. We see in Eq. (8) that every multiplication with tanh non-linearities \( (D_{\text{tanh}(\cdot)}) \), gating functions \( (D_{\sigma^l}(\cdot)) \), and with their derivatives can only ever decrease the singular values of \( W \), since all these 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 \( \alpha_t \) 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, which leads to \( h_t^1 \) and \( c_t^1 \) carrying 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
i^l_t$ from the architecture. We have empirically observed that the presence of the input gate does not significantly improve performance. Moreover, it actually harms training deeper networks. This empirical observation is in line with the results of [31], who show that removing the input and output gates does not greatly affect the performance of LSTMs.

More formally, our proposed STAR cell in the $l$-th layer takes the input $\hat{h}_{t-1}^l$ (in the first layer, $x_t$) at time $t$ and non-linearly projects it to the space where the hidden vector $h^l_t$ lives, equation 10. Furthermore, the previous hidden state and the new input are combined into the gating variable $k^l_t$ (equation 11). $k^l_t$ is our analogue of the forget gate and controls how information from the previous hidden state and the new input are combined into a new hidden state. The complete dynamics of the STAR unit is given by the expressions

\[
\begin{align*}
    z^l_t &= \tanh(W_z h_{t-1}^l + b_z) \\
    k^l_t &= \sigma(W_k h_{t-1}^l + W_h h_{t-1}^l + b_k) \\
    h^l_t &= \tanh((1 - k^l_t) \circ h_{t-1}^l + k^l_t \circ z^l_t).
\end{align*}
\]

The Jacobian matrices for the STAR cell can be computed similarly to how it is done for the vRNN and LSTM (see appendix). In this case each of the two Jacobians has average singular values equal to 0.5. This puts them between the LSTM, the LSTM with only a forget gate [31], the GRU, the RHN [38], 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). Classification performance is measured by the rate of correct predictions (top-1 accuracy). Throughout, we use 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.

5. Experiments

We evaluate the performance of several well-known RNN cells as well as that of the proposed STAR cell on five different sequence modelling tasks with four different datasets: sequential versions of MNIST [19], which are a common testbed for recurrent networks; two different remote-sensing datasets where time series of intensities observed in satellite images shall be classified into different agricultural crops [24, 25]; and, Jester [1] for hand gesture recognition. We use convolutional layers for gesture recognition and pixel-wise crop classification, whereas we employ conventional fully connected layers for the other tasks. The recurrent units we compare include the vRNN, the LSTM, the LSTM with only a forget gate [31], the GRU, the RHN [38], and the proposed STAR. The experimental results of [31], who show that removing the input and output networks. This empirical observation is in line with the re-

Figure 3: RNN cell structures: STAR, GRU and LSTM, respectively.
significantly more robust in that respect and can be trained up to a depth of 16 layers. On the comparatively easy and saturated MNIST data, the performance is comparable to a successfully trained LSTM (at depth 2-8 layers, LSTM training sometimes fails; the displayed accuracies are averaged only over successful training runs).

In 1 we show that our STAR cell outperforms most other state-of-the-art methods existing in literature. As STAR is specifically designed to improve gradient propagation in the vertical direction, we conduct one additional experiment with a hybrid architecture: we use LSTM with a forget gate (which achieves good performance on the MNIST dataset in the one layer case) as first layer of the network and we stack seven layers of STAR cells on top. This design increases the capacity of the first layer without endangering gradient propagation. This further improves accuracy for both MNIST and pMNIST, leading to on par performance across both tasks with the best state-of-the-art methods in dRNN [20] and BN-LSTM [10].

5.2. TUM time series classification

We evaluate model performance on a more realistic sequence modelling problem that aims at classifying agricultural crop types using sequences of satellite images. In this case, time-series modelling captures phenological evidence, i.e. different crops have different growing patterns.
### Table 1: Performance comparison for pixel-by-pixel MNIST tasks. Our best performing configuration **bold underlined**, top performers state-of-the-art **bold**.

| Method          | MNIST  | pMNIST | units |
|-----------------|--------|--------|-------|
| vRNN (1 layer)  | 24.3%  | 44.0%  | 128   |
| LSTM (4 layers) | **99.0%** | 91.5%  | 128   |
| iRNN [18]       | 97.0%  | 82.0%  | 100   |
| uRNN [2]        | 95.1%  | 91.4%  | 512   |
| FC uRNN [34]    | 96.9%  | 94.1%  | 512   |
| Soft ortho [33] | 94.1%  | 91.4%  | 128   |
| AntisymRNN [6]  | 98.8%  | 93.1%  | 128   |
| indRNN [20]     | **99.0%** | **96.0%** | 128   |
| BN-LSTM [10]    | **99.0%** | **95.4%** | 100   |
| sTANH-RNN [36]  | 98.1%  | 94.0%  | 128   |
| STAR (8 layers) | 99.2%  | 94.1%  | 128   |
| STAR (12 layers)| 99.2%  | 94.3%  | 128   |
| LSTM w/f STAR   | **99.3%** | **95.4%** | 128   |

---

over the season. The input is a time series of 26 multispectral 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 [24]. We use patches of 3×3 pixels recorded in 6 spectral channels and flattened into 54×1 vectors as input. The 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). STAR outperforms all baselines and its again more robust to stacking (Fig. 6). We also run an experiment for single step prediction (the model predicts a crop type after the entire sequence is presented.). STAR achieves 71.9% test accuracy against 70.5%, 70.4%, 70.2%, 70.1% and 68.8% accuracies obtained by GRU, RHN, LSTM w/f, LSTM and vRNN, respectively.

### 5.3. Convolutional RNN

We evaluate STAR on datasets where input data are image sequences and thus use convolutional layers. We analyse performance of STAR versus state-of-the-art on gesture recognition from video and pixel-wise crop classification.

**Hand-gesture recognition from video** The 20BN-Jester dataset V1 [1] is a large collection of densely-labelled 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. 6b. Going deeper improves the performance of all four tested convRNNs. The improvement is strongest for convolutional STAR, and the best performance is reached at high depth (12 layers). In summary, the results confirm both our intuitions 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 in the shallow 1-2 layer setting the conventional LSTM performs a slightly better than the three others, likely due to its larger capacity. Lastly, we conduct the same additional experiment with the hybrid architecture as we do for MNIST tasks. We stack seven layers of STAR on top of the one layer of LSTM. It further improves the results and achieves 92.7% accuracy (note that eight layers LSTM achieves 91.8% accuracy.).
Figure 7: Accuracy versus number of model parameters for the gesture recognition task (Jester).

Figure 8: Test accuracy versus training time for the gesture recognition task (Jester), 4 layers networks.

**TUM image series pixel-wise classification** We classify crops pixel-wise (and thus use convolutional layers) using a dataset [25] (TUM) containing Sentinel-2A optical satellite image sequences accompanied by ground-truth land cover maps. Each satellite image sequence contains 30 images of size $48 \times 48$ px collected in 2016 within a $102 \ km \times 42 \ km$ region north of Munich, Germany. Note that we only use those four image channels that have $10 \ m$ ground sampling distance. We compare pixel-wise classification accuracy for a network with a fixed depth of four layers and for four different basic recurrent cells LSTM, LSTM with only a forget gate, GRU, and the proposed STAR cell (Tab. 2). Moreover we include the performance obtained in [25] using a bidirectional convolutional GRU with a single layer. Our STAR cell outperforms all other methods (Tab. 2) while requiring less memory and being computationally less costly.

| Method          | Acc   | #params | #compute |
|-----------------|-------|---------|----------|
| bi-convGRU (1 layer)[25] | 89.7% | 6.2M    | 46bn     |
| convLSTM (4 layers)     | 90.6% | 292k    | 2.7bn    |
| convLSTM w/f (4 layers) | 89.6% | 161k    | 1.5bn    |
| convGRU (4 layers)      | 90.1% | 227k    | 2.1bn    |
| convSTAR (4 layers)     | **91.9%** | 124k    | **1.1bn** |

Table 2: Performance comparison for TUM pixel-wise image classification task.

5.4. Computational Resources and Training Time

We compare performance of our STAR cell with widely used recurrent units LSTM and GRU regarding parameter efficiency and training time for the convolutional version used in gesture recognition. We plot performance versus number of parameters (Fig. 7) STAR outperforms LSTM and performs on par with GRU, but requires only half the number of parameters. We plot accuracy on the validation dataset versus training time for different recurrent units for the gesture recognition task in Fig. 8. STAR does not only require significantly less parameters but can also be trained much faster: the validation accuracy on the dataset after 8 hours is comparable to the best validation achieved by the LSTM and the GRU after 20 hours of training.

6. Conclusion

We have proposed STAR, a novel stackable recurrent cell type that it is specifically designed to be employed in deep recurrent architectures. A thorough theoretical analysis and associated numerical simulations indicated that widely used standard RNN cells like LSTM and GRU do not preserve gradient magnitudes in the "vertical" direction during back-propagation. As the depth of the network grows, the risk of either exploding or vanishing gradients increases. We leveraged this analysis to design a novel cell that i) better preserves the gradient magnitude between two adjacent layers, ii) is better suited for deep architectures, and iii) requires less parameters than other widely used recurrent cells, such as LSTM and GRU. An extensive experimental evaluation on several publicly available datasets confirms that STAR units can be stacked into deeper architectures showing performance comparable to other state-of-the-art architectures. For future research it appears promising to investigate whether the analysis of the gradient flows could serve as a basis for better initialisation schemes. This could
help to compensate the systematic influences of the cell structure, e.g., gating functions, for training deep RNNs.

References

[1] The 20bn-jester dataset v1. https://20bn.com/datasets/jester.
[2] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In ICML, 2016.
[3] Yoshua Bengio et al. Learning deep architectures for AI.
[4] Yoshua Bengio et al. Learning long-term dependencies with gradient descent is difficult. IEEE TNN, 5(2):157–166, 1994.
[5] 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.
[6] Bo Chang, Minmin Chen, Eldad Haber, and Ed H Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In ICLR, 2019.
[7] Minmin Chen, Jeffrey Pennington, and Samuel S Schoenholz. Dynamical isometry and a mean field theory of rns: Gating enables signal propagation in recurrent neural networks. arXiv preprint arXiv:1806.05394, 2018.
[8] 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.
[9] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In ICML, 2015.
[10] T. Cooijmans, N. Ballas, C. Laurent, C. Gülcehre, and A. Courville. Recurrent batch normalization. In ICLR, 2017.
[11] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In ICASSP, 2013.
[12] Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. arXiv preprint arXiv:1602.06662, 2016.
[13] Sepp Hochreiter. Untersuchungen zu Dynamischen Neuronalen Netzen. Diploma Thesis, Technische Universität München, 91(1), 1991.
[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.
[15] 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.
[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In ICLR, 2014.
[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.
[18] 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.
[19] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. Proc. IEEE, 86(11):2278–2324, 1998.
[20] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In CVPR, 2018.
[21] Zhenyang Li, Kiriill Gavrilnyuk, Efstratios Gavves, Mihir Jain, and Cees GM Snoek. VideoSTM convolves, attends and flows for action recognition. CVIU, 166:41–50, 2018.
[22] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 2401–2409. JMLR. org, 2017.
[23] Sabeek Pradhan and Shayne Longpre. Exploring the depths of recurrent neural networks with stochastic residual learning, 2016.
[24] 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.
[25] Marc Rußwurm and Marco Körner. Multi-temporal land cover classification with sequential recurrent encoders. ISPRS International Journal of Geo-Information, 7(4):129, 2018.
[26] Amir Shahroudly, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+ d: A large scale dataset for 3d human activity analysis. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1010–1019, 2016.
[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.
[28] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. arXiv preprint arXiv:1505.00387, 2015.
[29] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In NIPS, 2014.
[30] Corentin Tallec and Yann Ollivier. Can recurrent neural networks warp time? In ICLR, 2018.
[31] Jos van der Westhuizen and Joan Lasenby. The unreasonable effectiveness of the forget gate. CoRR, abs/1804.04849, 2018.
[32] Oriol Vinyals and Quoc Le. A neural conversational model. arXiv preprint arXiv:1506.05869, 2015.
[33] Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. On orthogonality and learning recurrent networks with long term dependencies. In ICML, 2017.
[34] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In NIPS, 2016.
[35] Shi Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM networks: A machine learning approach for precipitation nowcasting. In NIPS, 2015.
[36] et al. Zhang, Saizheng. Architectural complexity measures of recurrent neural networks. In Advances in neural information processing systems, 2016.
[37] 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.

[38] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. In ICML, 2017.
A. RNN Cells Dynamics

In the following, we provide more detailed insights about the updating rules of the tested cell types.

Vanilla RNN update rule:

\[ h_t^l = \tanh(W_x h_{t-1}^l + W_h h_{t-1}^l + b) \]  
(13)

LSTM update rule:

\[ i_t^l = \sigma(W_x h_{t-1}^l + W_{hi} h_{t-1}^l + b_i) \]  
(14)

\[ f_t^l = \sigma(W_x h_{t-1}^l + W_{hf} h_{t-1}^l + b_f) \]  
(15)

\[ o_t^l = \sigma(W_x h_{t-1}^l + W_{ho} h_{t-1}^l + b_o) \]  
(16)

\[ z_t^l = \tanh(W_x h_{t-1}^l + W_{hz} h_{t-1}^l + b_z) \]  
(17)

\[ c_t^l = f_t^l \circ c_{t-1}^l + i_t^l \circ z_t^l \]  
(18)

\[ h_t^l = o_t^l \circ \tanh(c_t^l) \]  
(19)

LSTM with only forget gate update rule:

\[ f_t^l = \sigma(W_x h_{t-1}^l + W_{hf} h_{t-1}^l + b_f) \]  
(20)

\[ z_t^l = \tanh(W_x h_{t-1}^l + W_{hz} h_{t-1}^l + b_z) \]  
(21)

\[ h_t^l = \tanh(f_t^l \circ h_{t-1}^l + (1 - f_t^l) \circ z_t^l) \]  
(22)

GRU update rule:

\[ z_t^l = \sigma(W_x h_{t-1}^l + W_{hz} h_{t-1}^l + b_z) \]  
(23)

\[ r_t^l = \sigma(W_x h_{t-1}^l + W_{hr} h_{t-1}^l + b_r) \]  
(24)

\[ h_t^l = (1-z_t^l)\circ h_{t-1}^l + z_t^l \circ \tanh(W_x h_{t-1}^l + W_{hh}(r_t^l \circ h_{t-1}^l) + b_h) \]  
(25)

STAR Jacobians: The expressions for the Jacobians of STAR cell are given in Eq. (26)-(27).

Convolutional STAR: We briefly describe the convolutional version of our proposed cell. The main difference is that matrix multiplications now become convolutional operations. The dynamics of the convSTAR cell is given in the following equations.

\[ K_t^l = \sigma(W_x * h_{t-1}^l + W_h * h_{t-1}^l + B_K) \]  
(28)

\[ Z_t^l = \tanh(W_x * h_{t-1}^l + B_z) \]  
(29)

\[ H_t^l = \tanh(h_{t-1}^l + K_t^l \circ (Z_t^l - h_{t-1}^l)) \]  
(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 [8] and the LSTM with only forget gate for the synthetic dataset (Sec. B.1); and for the real dataset, MNIST (Sec. B.2).

B.1. Synthetic Dataset

The setup of the numerical simulations is the same as the one described in Section 3. As can be seen from Fig. 9 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. 10. 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 9: Mean gradient magnitude w.r.t. the parameters for LSTM with only forget gate, GRU, and the proposed STAR cell. top row: loss L(h_T) only on final prediction. bottom row: loss L(h_T, ..., h_T) over all time steps.](image)

B.2. MNIST Dataset

In this section, we perform the same numerical analysis conducted before but using MNIST as input data. The goal is to verify whether during the first epoch the gradient propagation behaves in the same way as for the synthetic dataset. First, in Fig 14 and Fig 11, we plot the evolution of the Hilbert-Schmidt norm (also called Frobenius norm) normalized by the square root of the hidden state size and the average hidden state value, respectively. The experiments are conducted using the proposed STAR method with
\[
J_t^1 = D_{\tanh(h_{t-1}^l+k_t^l \circ (z_t^l-h_{t-1}^l))}(D_{z_t^l-h_{t-1}^l} D(k_t^l) W_x + D_{k_t^l} D(z_t^l) W_z)
\]

\[
H_t^1 = D_{\tanh(h_{t-1}^l+k_t^l \circ (z_t^l-h_{t-1}^l))}(I + D_{z_t^l-h_{t-1}^l} D(k_t^l) W_h - D_{k_t^l})
\]

Figure 10: Mean-normalised standard deviation of gradient magnitude for LSTM with only forget gate, GRU, and the proposed STAR cell. top row: loss $\mathcal{L}(h_{t+1}^l)$ only on final prediction. bottom row: loss $\mathcal{L}(h_{1}^l \ldots h_{T}^l)$ over all time steps.

C.1. Pixel-by-pixel MNIST

Following [30], chrono initialisation is applied for the bias term of $k$, $b_k$. The basic idea is that $k$ should not be too large; such that the memory $h$ 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 [16] with learning rate 0.001, $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

C.2. TUM time series classification

We use the same training procedure as described in the previous section for pixel-by-pixel MNIST. 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 [16] with learning rate 0.001 and $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

C.3. Hand-gesture recognition from video

All convolutional kernels are of size $3 \times 3$. 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 trained 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. $L_2$-regularisation with weight 0.00005 is applied to all parameters.

C.4. TUM image series pixel-wise classification

All convolutional kernels are of size $3 \times 3$. Each convolutional RNN layer has 32 filters. A shallow CNN is used to convert the hidden state to a label, with 2 layers that have filter depths 64. All models are fitted with Adam [16]. The batch size is set to 1, the learning rate starts at 0.001 and decays polynomially to 0.000001 over a total of 25 epochs.

C. Training details

We provide more details about training procedures for the experimental analysis in the main paper in this section.
Figure 11: Mean hidden state vector, $E_{t,n}[h^t]$ of pix-by-pix MNIST during 1st epoch. Different curves correspond different layers.

Figure 12: Mean gradient magnitude w.r.t. the parameters for vRNN, LSTM, GRU, and the proposed STAR cell for MNIST dataset. Loss $L(h^T_f)$ only on final prediction.

Figure 13: Gradient magnitude comparison within a single run for MNIST dataset. top row: GRU samples, bottom row: STAR samples. Samples are randomly picked.

Figure 14: Weight matrix norms of pix-by-pix MNIST during 1st epoch, the Hilbert-Schmidt norm, $\|A\|_{HS} = \sqrt{Tr(AA^T)}$, divided by $\sqrt{m}$. Different curves correspond different layers.