The streaming rollout of deep networks - towards fully model-parallel execution

Volker Fischer  
Robert Bosch Center for Artificial Intelligence  
Robert Bosch GmbH  
Stuttgart  
volker.fischer@de.bosch.com

Jan Köhler  
Robert Bosch Center for Artificial Intelligence  
Robert Bosch GmbH  
Stuttgart  
jan.koehler@de.bosch.com

Thomas Pfeil  
Robert Bosch Center for Artificial Intelligence  
Robert Bosch GmbH  
Stuttgart  
thomas.pfeil@de.bosch.com

Abstract

Deep neural networks, and in particular recurrent networks, are promising candidates to control autonomous agents that interact in real-time with the physical world. However, this requires a seamless integration of temporal features into the network’s architecture. For the training of and inference with recurrent neural networks, they are usually rolled out over time, and different rollouts exist. Conventionally, during inference the layers of a network are computed in a sequential manner resulting in sparse temporal integration of information and long response times. In this study, we present a theoretical framework to describe the set of all rollouts and demonstrate their differences in solving specific tasks. We prove that certain rollouts, also with only skip and no recurrent connections, enable earlier and more frequent responses, and show empirically that these early responses have better performance. The streaming rollout maximizes these properties and, in addition, enables a fully parallel execution of the network reducing the runtime on massively parallel devices. Additionally, we provide an open-source toolbox to design, train, evaluate, and online-interact with streaming rollouts.

1 Introduction

Over the last years, the combination of newly available large datasets, parallel computing power, and new techniques to implement and train deep neural networks has led to significant improvements in the fields of vision [1], speech [2], and reinforcement learning [3]. In the context of applications, neural networks usually interact with the physical world in real-time which renders it essential to integrate the processing of temporal information into the network’s design.

Recurrent neural networks (RNNs) are one common approach to leverage temporal context and have gained increasing interest not only for speech [4] but also for vision tasks [5]. For RNNs some neural activations are used to inform future computations of the same or preceding neuron activations, hence introducing a recursive dependency between neuron activations. This augments the network with a memory mechanism and allows it, unlike feed-forward neural networks, to exhibit dynamic temporal behavior on a stream or sequence of inputs. For training and inference, backpropagation through time (BPTT) [6] or its truncated version [6, 7] are used, where the RNN is rolled out (or unrolled) through
time disentangling the recursive dependencies and hence transforming the recurrent network into a feed-forward network.

Since unrolling a cyclic graph is not well-defined \[8\], different possible rollouts exist for the same neural network (Figure 1). This is due to the rollout-process itself, as there are several ways to unroll cycles with length greater than 2. More general, there are two ways to unroll every edge: having the edge connect its source and target nodes at the same point in time (see e.g. vertical edges in Figure 1b) or bridging time steps (see e.g. Figure 1c). One contribution of this study is to prove that the number of different rollouts increases exponentially with network complexity. Bridging is especially necessary for self-recurrent edges or larger cycles in the network, so that the rollout in fact becomes a feed-forward network. Conventionally, in a rollout most edges are applied in the intra-frame non-bridging manner and only bridge time steps if necessary \[9\][10][11][12], which we call sequential rollouts throughout this work.

The central observation of this work is that different rollouts induce different levels of model-parallelism for the unrolled network. In rollouts inducing complete model-parallelism, which we call streaming, nodes of a certain time step in the unrolled network become computationally disentangled and can, hence, be computed in parallel. Further, this idea is not restricted to recurrent networks, but generalizes to a large variety of network architectures covered by the presented graph-theoretical framework in Section 3. In Section 4 we show experimental results that emphasize the difference of rollouts for both networks with recurrent and skip and only skip connections. In this study we are not concerned comparing performances between networks, but between different rollouts of a given network.

Our theoretical and empirical findings show that streaming rollouts enable fully model-parallel inference achieving low-latency and high-frequency responses. These features are particularly important for real-time applications such as autonomous cars \[13\], chat bots \[14\] or UAV systems \[15\] in which the neural networks have to make complex decisions on high dimensional and frequent input signals within a short time.

To the best of our knowledge, up to this study, no general theory or experimental investigations exist that compare different rollouts and our contributions can be summarized as follows:

- We provide a theoretical framework to describe rollouts of deep neural networks and show that, and in some cases how, different rollouts lead to different network behavior.
- We formally introduce streaming rollouts enabling fully model-parallel network execution, and mathematically prove that streaming rollouts have the shortest response time to and highest sampling frequency of inputs.
- We empirically give examples underlining the theoretical statements and showing that streaming rollouts can further outperform other rollouts by yielding better early and late performance.
- We provide an open-source toolbox specifically designed for streaming rollouts of deep neural networks.

2 Related work

The idea of RNNs dates back to the mid-70s \[16\] and was popularized by \[17\]. RNNs and their variants, especially long short-term memory networks (LSTM) \[18\], considerably improved performance in different domains such as speech recognition \[4\], handwriting recognition \[5\], machine translation \[19\], optical character recognition (OCR) \[20\], text-to-speech synthesis \[21\], social signal classification \[22\] or online multi-target tracking \[23\]. The review \[24\] gives an overview of the history of and benchmark records set by DNNs and RNNs.
We give an explicit formulation of this abstraction for deep neural networks in Section A1.2, which we describe dependencies inside a neural network \( N \) we also use for our experiments. Important concepts introduced here are illustrated in Figure A2.

**Variants of RNNs:** Several variants of RNN architectures exist having different mechanisms to memorize and integrate temporal information. Long Short-Term Memory (LSTM) networks [18] and related architectures like Gated Recurrent Unit (GRU) networks [25] or recurrent highway networks [26] exist. Neural Turing Machines (NTM) [27] like Differentiable Neural Computers (DNC) [28] extend RNNs by an addressable external memory.

Bi-directional RNNs (BRNNs) [29] incorporate the ability to model the dependency on future information. Numerous works extend and improve these RNN variants creating architectures with advantages for training or certain data domains [e.g. [30] [31] [32] [33]].

**Response time:** While RNNs are the main reason to use network rollouts, in this work we also investigate rollouts for non-recurrent networks. Theoretical and experimental results suggest that different rollout types yield different behavior especially for networks containing skip connections. The rollout pattern influences the response time of a network which is the duration between input (stimulus) onset and network output (response). The runtime of inference and training for the same network can, e.g., be reduced by network compression [34] [35] or computational implementation optimization [36] [37].

Shortcut or skip connections also play an important role to decrease response times. Shortcut branches attached to intermediate layers allow earlier predictions (e.g. BranchyNet [38]) and iterative predictions refine from early and coarse to late and fine class predictions (e.g. feedback networks [12]). In [39] the authors show that identity skip connections, as used in Residual Networks (ResNet) [11], can be interpreted as local network rollouts acting as filters, which could also be achieved through recurrent self-connections. The good performance of ResNets underlines the importance of local recurrent filters.

**Rollouts:** To train RNNs, different rollout patterns are applied in the literature, though lacking a theoretically founded background. One of the first to describe the transformation of a recurrent MLP into an equivalent feed-forward network and depicting it in a streaming rollout fashion was [40] ch. 9.4. The most common way in literature to unroll networks over time is to duplicate the model for each time step as depicted in Figure 1b [ch. 10.1 in [41], 9, 10, 11, 12]. However, as we will show in this work, this rollout pattern is neither the only way to unroll a network nor the most efficient.

### 3 Graph representations of network rollouts

We describe dependencies inside a neural network \( N \) as a directed graph \( N = (V, E) \). The nodes \( v \in V \) represent different layers and the edges \( e \in E \subset V \times V \) represent transformations introducing direct dependencies between layers. We will allow self-connections \( (v, v) \in E \) and larger cycles in a network. Before stating the central definitions and propositions, we introduce notations used throughout this section and for the proofs in the appendix.

Let \( G = (V, E) \) be a directed graph with vertices (or nodes) \( v \in V \) and edges \( e = (e_{src}, e_{tgt}) \in E \subset V \times V \). Since neural networks process input data, we hence denote the input of the graph as set \( I_G \), consisting of all nodes without incoming edge:

\[
I_G = \{ v \in V : \exists u \in V : (u, v) \in E \}.
\]

A **path** in \( G \) is a mapping \( p : \{1,\ldots,L\} \rightarrow E \) with \( p(i)_{tgt} = p(i+1)_{src} \) for \( i \in \{1,\ldots,L-1\} \) where \( L \in \mathbb{N} \) is the **length** of \( p \). We denote the length of a path \( p \) as \( |p| \) and the number of elements in a set \( A \) as \( |A| \). A path \( p \) is called **loop** or cycle if \( p(|p|)_{tgt} = p(1)_{src} \) and it is called **minimal iff** \( p \) is injective. The set of all cycles is denoted as \( C_G \). Two paths are called **non-overlapping** if they share no edges. We say a graph is **input-connected** if for every node \( v \) exists a path \( p \) with \( p(|p|)_{tgt} = v \) and \( p(1)_{src} \in I_G \). Now we proceed with our definition of a (neural) network.

**Definition (network):** A **network** is a directed and input-connected graph \( N = (V, E) \) for which \( 0 < |E| < \infty \) and \( I_N \neq \emptyset \).

For our claims, this abstract formulation is sufficient and, while excluding certain artificial cases, it ensures that a huge variety of neural network types is covered (see Figure A1 for network examples). We give an explicit formulation of this abstraction for deep neural networks in Section A1.2 which we also use for our experiments. Important concepts introduced here are illustrated in Figure A2. In this work we separate the concept of network rollouts into two parts: The **rollout pattern** and its associated **rollout windows** (e.g. \( W = 1 \) for Figure 1 and \( W = 2 \) for Figure 2):
The previous lemma shows that the number of valid rollout patterns increases exponentially with network complexity. Inference of a rollout window is conducted in a sequential manner. This means the state of all nodes in the rollout window is successively computed depending on the availability of already computed source nodes\footnote{given the state of all input nodes at all frames and initial states for all nodes at the zero-th frame}. The chosen rollout pattern determines the mathematical function...
this rollout represents, which may be different between rollouts, e.g. for skip connections. In addition the chosen rollout also determines the order in which nodes can be computed leading to different runtimes to compute the full state of a rollout window.

We now introduce tools to compare these addressed differences between rollouts. States of the rollout window encode which nodes have been computed so far and update steps, based on the previous state, determine the next state. Update-tableaus list after how many update steps nodes in the rollout window are computed. Update states, update steps, and update tableaus are shown for an example network in Figure 2 and in Figure A2.

**Definition (update state, update step, tableau, and factor):** Let \( R \) be a valid rollout pattern of a network \( N = (V, E) \). A state of the rollout window \( R_W \) is any mapping \( S : V_W \to \{0, 1\} \). Let \( \Sigma_W \) denote the set of all states. We define the full state \( S_{\text{full}} \) and initial state \( S_{\text{init}} \) as:

\[
S_{\text{full}} = 1 \quad S_{\text{init}}((i, v)) = 1 \iff v \in I_N \land i = 0.
\]

Further, we define the inference update-step \( U \) which updates states \( S \). Because the updated state \( U(S) \) is again a state and hence a mapping, we define \( U \) by specifying the mapping \( U(S) \):

\[
U : \Sigma_W \to \Sigma_W \quad U(S) : V_W \to \{0, 1\}
\]

\[
U(S)(v) = \begin{cases} 
1 & \text{if } S(v) = 1 \text{ or if for all } (\pi, v) \in E_W : S(\pi) = 1 \\
0 & \text{otherwise}
\end{cases}
\]

We call the mapping \( T : V_W \to N \) the inference tableau:

\[
T(\pi) = \max_{p \in P_\pi} |p| = \arg\min_{n \in \mathbb{N}} \{U^n(S_{\text{init}})(\pi) = 1\}
\]

where \( U^n \) is the \( n \)-th recursive application of \( U \) and for \( \pi \in V_W \), \( P_\pi \) denotes the set of all paths in \( R_W \) that end at \( v \), \( p(|p|_{\text{last}}) = v \) and for which their first edge may start but not end in the 0-th frame, \( p(1)_{\text{last}} \notin \{0\} \times V \). Hereby we exclude edges (computational dependencies) which never have to be computed, because all nodes in the 0-th frame are initialized from start.

Dependencies of \( U \) and \( T \) on the rollout window \( R_W \) were dropped in the notation, and if needed will be express as \( U_{R_W} \) and \( T_{R_W} \). Further, we call the maximal value of \( T \) over the rollout window of size 1 the rollout pattern’s inference factor:

\[
F(R) := \max_{\pi \in V_1} T_{R_1}(\pi).
\]

**Proof:** In Section A1.6 we prove Equation (2).

In practice inference with rollout windows always starts with the initial state \( S_{\text{init}} \) (compare Figure 2 for first state right from network). Successive applications of the update step \( U \) updates all nodes until the fully updated state is reached \( S_{\text{full}} \) (see Section A1.5 for a proof and Figure 2).

In the following, time is measured by the number of performed update steps. This is not well-defined, because the execution time of an update step depends on the state \( S \) to be updated and the rollout pattern, e.g., which and how many nodes can be updated at once. For now we will assume independence, but will address this issue in Section 5.

**Theorem:** Let \( R \) be a valid rollout pattern for a network \( N = (V, E) \) then the following statements are equivalent:

a) \( R \) and the streaming rollout pattern \( R_{\text{stream}} \) are equally model-parallel.

b) The first frame is updated entirely after the first update step: \( F(R) = 1 \).

c) For \( W \in \mathbb{N} \), the \( i \)-th frame of \( R_W \) is updated at the \( i \)-th update step:

\[
\forall \pi = (i, v) \in V_W : T(\pi) \leq i.
\]

d) For \( W \in \mathbb{N} \), the inference tableau of \( R_W \) is minimal everywhere and over all rollout patterns. In other words, responses are earliest and most frequent:

\[
\forall \pi \in V_W : T_{R_W}(\pi) = \min_{R \in \mathbb{R}_W} T_{R_W}(\pi).
\]

**Proof:** See appendix Section A1.7.
Figure 3: Classification accuracy for sequential and streaming rollouts on MNIST, CIFAR10 and GTRSB (for networks and data see Figures 1, 3d, A3 and A4). a-c: Average classification results on MNIST over computation time measured by the number of update steps of networks with skip and recurrent (SR), with skip (S) and only feed-forward (FF) connections. d: The input (top row) is composed of digits (bottom row) and noise (middle row). Note that the input is aligned to the time axis in (a). Red diamonds and blue stars indicate inputs sampled by streaming and sequential rollouts, respectively. e: Classification results of the network DSR2 on CIFAR10. f: Accuracies at the time of first output of sequential networks (see (V) in e) over network depth. g: Average classification results on GTRSB sequences starting at index 1 of the original sequences. h: Final classification accuracies (see (VI) in f) over the start index of the input sequence. Standard errors of the means are plotted in all subfigures, but in (a-c) they are too small to be visible, and for (e and f) only single trials are used.

4 Experiments

To demonstrate the significance of the chosen rollouts w.r.t. the runtime for inference and achieved accuracy, we compare the two extreme rollouts, the most parallelizable, i.e. streaming ($R \equiv 1$, results in red in Figure 3), and the most sequential rollout ($R(e) = 0$ for maximal number of edges, results in blue in Figure 3). We assume that whatever can be calculated in parallel is calculated in parallel and that time can be measured in the number of update steps (for a discussion see Section 5). In all experiments, we consider a response time task in which the input is a sequence of images and the networks have to respond as quickly as possible with the correct class. We want to restate, that we do not compare performances between networks but between rollout patterns of the same network.

Datasets: We evaluated rollouts on three datasets: MNIST [42], CIFAR10 [43] and the German traffic sign recognition benchmark (GTRSB) [44]. Since, for these datasets, single images are already sufficient for almost perfect classification results, we add noise to images (see Figure 3d and Figure A3b, c), thus, temporal integration is necessary for good performance. In case of GTRSB this noise can, e.g., be seen as noise induced by the sensor as predominant under poor lighting conditions. GTRSB contains tracks with 30 frames each.

Networks: We compared behaviors of streaming and sequential rollout patterns on MNIST for three variants of networks with two hidden layers (see Figure 1 and Figure A4). For evaluation on CIFAR10, we generated a sequence of incrementally deeper networks (DSR0, ..., DSR6) by adding more and more layers to the blocks of a recurrent network with skip connections in a dense fashion (details in

---

2The most sequential rollout is unique here since our networks do not contain top-down connections. For sequential rollouts that are ambiguous see Figure 2.
For evaluation on GTRSB we used DSR4 leaving out the recurrent connection. Details about data, preprocessing, network architectures and training process are in Section A2.

Results: To compare different rollouts we primarily investigated their test accuracies over the duration needed to compute these results (Figure 3a, c, e and g). Due to the parallel computation of all nodes in a frame in the streaming case, the sampling frequency of input images (every time step; see red diamonds in Figure 3d) is always equal to or higher than for sequential rollouts (see d in main theorem in Section 3). In contrast, the sampling frequency of sequential rollouts decreases linearly with the length of the longest path (every third image; see blue stars in Figure 3d; note that the linear scaling of the abscissa changes at the vertical dashed line).

In the following three paragraphs, we show the difference between streaming and sequential rollouts with increasingly complex networks on the MNIST dataset. In the case of neither recurrent, nor skip connections, the streaming rollout is mathematically identical to the sequential rollout. None of the rollouts can integrate information of inputs over time and, consequently, both rollouts perform classification on single input images with the same response time (Figure 3c). Note that data points are similar and partly overlap. However, due to the pipelined structure of computations in the streaming case, outputs are more frequent.

For networks with skip, but without recurrent connections the difference between streaming and sequential rollouts can be shown best. While the sequential rollout can only perform classification on single images, the streaming rollout is capable to integrate over several input images due to the skip connections that bridge time (see Figure 3b; note that here the skip connection allows to integrate information over two inputs).

In the streaming case, skip connections cause shallow shortcuts in time that can result in earlier, but initially worse performance than for deep sequential rollouts (see I in Figure 3a). The streaming rollout responds 1 update step earlier than the sequential rollout since its shortest path is shorter by 1 (for shortest paths see Figure 1). These early first estimations are later refined when longer paths and finally the longest path from input to output contributes to classification. For example, after 3 time steps in Figure 3a, the streaming rollout uses the full network, i.e. the longest and skipping paths, and only the shortcut, i.e. the skipping path, to integrate the images presented before the first and second update step (index 0 and 1), respectively, into classification. In contrast, the sequential rollout could not integrate images over time, yet, and only uses the image presented before the first update step (index 0) for classification (compare blue to red arrows connecting Figure 3d and a).

High sampling frequencies and shallow shortcuts via skip connections establish a high degree of temporal integration early on and result in better early performance (see III in Figure 3a). In the long run, however, classification performances are comparable between streaming and sequential rollouts and the same number of input images is integrated over (see IV in Figure 3a).

We repeated similar experiments for the CIFAR10 dataset to demonstrate the increasing advantages of the streaming over sequential rollouts for deeper and more complex networks. For the network DSR2 with the shortest path of length 4 and longest path of length 6 the first response of the streaming rollouts is 2 update steps earlier than for sequential rollouts and shows better early performance (see V in Figure 3a). Due to the dense structure of skip connections within the network, the refinement of results is also more finely grained compared to the MNIST example. With increasing depth (length of the longest path) over the sequence of networks DSR0, ..., DSR6 the time to first response stays constant for streaming, but linearly grows with the depth for sequential rollouts (see Section 3d). The difference of early performance (see V in Figure 3a) widens with deeper networks (Figure 3a).

For the evaluation of rollouts on GTRSB data we consider the DSR4 network. Self-recurrence is omitted since the required short response times of this task cannot be achieved with sequential rollouts due to the very small sampling frequencies. Consequently, for a fair comparison, we calculate the classifications of the first 8 images in parallel for the sequential case. In this case where both rollouts use the same amount of computations, performance for the sequential rollout increases over time due to less blurry input images, while the streaming rollout in addition performs temporal integration over input images using the skip connections and yielding better performance (see VI in Figure 3a). This results in better performance of streaming compared to sequential rollouts for more distant objects (Figure 3b) marking major improvements in the regime of input space that is most relevant for safety in driving assistance systems.
The presented theory for network rollouts is generically applicable to a vast variety of deep neural networks and is not constrained to recurrent network but could be used on forward (e.g. VGG [45], AlexNet [46]) or skipping networks (e.g. GoogLeNet [47], ResNet [1], DenseNet [48]). We restricted rollout patterns to have values $R(e) \in \{0, 1\}$ and did neither allow edges to bridge more than 1 frame ($R(e) > 1$) nor pointing backwards in time $R(e) < 0$. The first case is subsumed under the presented theory using copy-nodes to realize longer forward connections, where in the second case rollouts with backward connections loose the real-time capability, because information from future frames would be used.

One disadvantage of the streaming rollout pattern seems to be that the deeper the networks the longer their rollout windows are. Rollout windows are as long as the longest minimal path connecting input to output, i.e. all paths have appeared at least once in the rollout window. For sequential rollout patterns this is not the case, since, e.g., for a feed-forward network the longest minimal path is already contained in the very first frame. In general, for streaming rollouts, we propose to use very shallow rollouts (e.g. $W = 1$), using one rollout window as initialization for the next, enabling a potentially infinite memory for recurrent networks.

In addition to faster responses to inputs, also the overall runtime for the calculation of a full rollout is reduced for streaming compared to other, especially sequential, rollouts. Throughout this study, we measured runtime by the number of update steps and their duration. Assuming a fully parallel hardware, streaming rollouts manifest the best case scenario in terms of maximal parallelization and the inference of a rollout window of size $W$ would take $W$ times the runtime of the computationally most expensive node update. However, sequential rollouts would not benefit from the assumed parallelism of the assumed hardware due to in-frame dependencies of computations that cause nodes to be computed one after the other.

This difference in runtime depends on the hardware used for execution. Although commonly used GPUs provide sufficient parallelism to significantly speed up the calculations of activations within a layer, they are not parallel enough to enable the parallel computation of multiple layers. Hence, streaming and sequential rollouts have approximately the same runtime on GPUs. However, novel massively parallel hardware architectures like the TrueNorth chip [49, 50] allow to store and run the full network rollouts on chip reducing the runtime of rollouts drastically and therefore making streaming rollouts highly attractive for training and inference. The limited access to massively parallel hardware may be one reason, why streaming rollouts have not been thoroughly discussed, yet.

Furthermore, not only the hardware, but also the software frameworks must support the parallelization of independent nodes in their computation graph to exploit the advantages of streaming rollouts. This is usually not the case and by default sequential rollouts are used. Upon publication of this study we publically release both a framework dedicated to streaming rollouts (see Section A3) and the source code to reproduce our results.

In scenarios in which short reaction times are necessary, a steady-state approximation of the physical world has to be established. This is only achievable by comparably high sampling frequencies as enabled by the streaming rollout. Spiking neural networks potentially have even higher sampling frequencies and also process information in a streaming fashion, but on a more fine grained level. Instead of block-wise and synchronous computations on layer level in streaming rollouts, spiking networks process information with independent and asynchronous neurons.

One direction of future work is the investigation of streaming rollouts on more complex tasks, e.g. by their combination with more sophisticated memory mechanisms such as used in NTMs. The contribution of this work is primarily the introduction of a theoretical concept and experiments are used to underline, illustrate and to some degree extend this concept. Another interesting research area is the exploration of mechanisms to guide and control information flow in networks, e.g. through gated skips (bottom-up) and recurrent (top-down) connections. On one side, new sensory information should be distributed quickly into deeper layers. On the other side, high-level representations and knowledge of the network about its current task should stabilize, predict and constrain (e.g. compare to attention) lower-level representations. Also existing techniques for network training could be optimized with regard to the regime of streaming rollouts.
**Conclusion:** We presented a theoretical framework for network rollouts and investigated the streaming rollout, which fully disentangles computational dependencies between nodes and hence enables full model-parallel inference. Furthermore, we empirically demonstrated its superiority over non-streaming rollouts for different image data sets due to faster first responses to and higher sampling of inputs. Along with a decreased overall runtime we hope that our work will encourage the scientific community to leverage and further study the advantages of streaming rollouts in preparation to future massively parallel hardware systems.
References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2016.

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In International Conference on Machine Learning (ICML), pages 173–182, 2016.

[3] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In International Conference on Machine Learning (ICML), pages 1329–1338, 2016.

[4] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. In International Conference on Artificial Neural Networks, pages 220–229, 2007.

[5] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In Advances in Neural Information Processing Systems (NIPS), pages 545–552, 2009.

[6] Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. Neural networks, 1(4):339–356, 1988.

[7] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. Backpropagation: Theory, architectures, and applications, 1:433–486, 1995.

[8] Qianli Liao and Tomaso Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. arXiv:1604.03640, 2016.

[9] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In International Conference on Machine Learning (ICML), pages 1310–1318, 2013.

[10] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3367–3375, 2015.

[11] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging. arXiv:1508.01991, 2015.

[12] A. R. Zamir, T.-L. Wu, L. Sun, W. Shen, B. E. Shi, J. Malik, and S. Savarese. Feedback Networks. arXiv 1612.09508, 2016.

[13] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3530–3538, 2017.

[14] Oriol Vinyals and Quoc Le. A neural conversational model. In International Conference on Machine Learning (ICML) Deep Learning Workshop, 2015.

[15] Chih-Min Lin, Ching-Fu Tai, and Chang-Chih Chung. Intelligent control system design for uav using a recurrent wavelet neural network. Neural Computing & Applications, 24(2):487 – 496, 2014.

[16] William A Little. The existence of persistent states in the brain. In From High-Temperature Superconductivity to Microminiature Refrigeration, pages 145–164. Springer, 1974.

[17] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the national academy of sciences, 79(8):2554–2558, 1982.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[19] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems (NIPS), pages 3104–3112, 2014.

[20] Thomas M Breuel, Adnan Ul-Hasan, Mayce Ali Al-Azawi, and Faisal Shafait. High-performance OCR for printed English and Fraktur using LSTM networks. In International Conference on Document Analysis and Recognition (ICDAR), pages 683–687, 2013.
[21] Yuchen Fan, Yao Qian, Feng-Long Xie, and Frank K Soong. TTS synthesis with bidirectional LSTM based recurrent neural networks. In Fifteenth Annual Conference of the International Speech Communication Association, 2014.

[22] Raymond Brueckner and Bjorn Schulter. Social signal classification using deep BLSTM recurrent neural networks. In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 4823–4827, 2014.

[23] Anton Milan, Seyed Hamid Rezatofighi, Anthony R Dick, Ian D Reid, and Konrad Schindler. Online multi-target tracking using recurrent neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pages 4225–4232, 2017.

[24] Jürgen Schmidhuber. Deep learning in neural networks: An overview. Neural networks, 61:85–117, 2015.

[25] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. arXiv:1409.1259, 2014.

[26] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutnik, and Jürgen Schmidhuber. Recurrent highway networks. arXiv:1607.03474, 2016.

[27] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. arXiv:1410.5401, 2014.

[28] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. Nature, 538:471–476, 2016.

[29] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11):2673–2681, 1997.

[30] 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 International Conference on Learning Representations (ICLR), 2018.

[31] Golan Pundak and Tara N Sainath. Highway-LSTM and recurrent highway networks for speech recognition. In Proceedings of Interspeech, 2017.

[32] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. arXiv:1312.6026, 2013.

[33] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. In International Conference on Learning Representations (ICLR), 2017.

[34] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In International Conference on Learning Representations (ICLR), 2016.

[35] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. arXiv:1511.06530, 2015.

[36] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 4013–4021, 2016.

[37] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. In International Conference on Learning Representations (ICLR), 2014.

[38] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In International Conference on Pattern Recognition (ICPR), 2016.

[39] Klaus Greff, Rupesh K. Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. In International Conference on Learning Representations (ICLR), 2017.

[40] Marvin Minsky and Seymour A. Papert. Perceptrons: An introduction to computational geometry. MIT press, 1969. retrieved from the 1988 reissue.

[41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press Cambridge, 2016.

[42] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.
[43] Alex Krizhevsky and Geoffrey E. Hinton. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 2009.

[44] Johannes Stallkamp, Marc Schlippsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1453–1460. IEEE, 2011.

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[47] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[48] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[49] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taban Armon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[50] Steven K. Esser, Paul A. Merolla, John V. Arthur, Andrew S. Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J. Berg, Jeffrey L. McKinstry, Timothy Melano, Davis R. Barch, Carmelo di Nolfo, Pallab Datta, Armon Amir, Brian Taban Myron D. Flickner, and Dharmendra S. Modha. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, 113(41):11441–11446, 2016.

[51] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning 4*, 2012.

[52] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. arXiv:1605.02688, 2016.

[53] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/ Software available from tensorflow.org.

[54] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.
The streaming rollout of deep networks - towards fully model-parallel execution
Supplementary material

A1 Proofs and notes for theory chapter

To improve readability, we will restate certain parts of the theory chapter from above.

A1.1 Illustrative example of central theoretical concepts from Section 3

![Illustrative example of central theoretical concepts from Section 3]

Figure A1: Examples of networks covered by the presented theory in Section 3. The crossed out network has no input and is consequently not valid.

Concepts from Section 3 are illustrated in Figure A2. An exemplary network with a skip and a recurrent connection is unrolled into three different rollout patterns.

![Illustrative example for central concepts]

Figure A2: Illustrative example for central concepts. From left to right are shown: an example of a network $N$, three different of its rollout patterns $R^I, R^{II}, R^{III}$, the rollout windows of size $W = 3$ for the three rollout patterns, the values of the initial state $S_{\text{init}}$ (which is the same) on the rollout windows nodes $v \in V_3$ (white: $S_{\text{init}}(v) = 0$, black: $S_{\text{init}}(v) = 1$), the state after the first update step $U(S_{\text{init}})$, and finally the values of the inference update tableau $T(v)$ for all $v \in V_3$.

A1.2 Note: Connecting the theory of networks with deep neural networks

For deep networks, nodes $v$ correspond to layers and edges $e$ to transformations between layers, such as convolutions (e.g. see networks in Figure A4 and Figure A1). To a node $v$ a state $x_v \in \mathbb{R}^{D_v}$ is assigned, e.g. an image with $D_v = 32 \times 32 \times 3$. Let denote $y_e$ the result of the transformation $f_e$ which is specified by an edge $e = (u, v)$:

$$y_e = f_e(\theta_e, x_u),$$
where \( \theta_e \) are parameters of the edge \( e \), e.g. a weight kernel.

For the node \( v \), let \( \text{SRC}_v \) denote the set of all edges targeting \( v \). Ignoring the temporal dimension, a node’s state is then computed as:

\[
x_v = f_v(\vartheta_v, y_{e_1}, \ldots, y_{e_{|\text{SRC}_v|}}),
\]

where \( \vartheta_v \) are parameters of the vertex \( v \) (e.g. biases), and the mapping \( f_v \), specifying how the sources are combined (e.g. addition and/or multiplication). For the experiments in this work, for every node \( v \) all results of incoming transformations were summed up:

\[
x_v = f_v(b, y_{e_1}, \ldots, y_{e_{|\text{SRC}_v|}}) = \sigma \left( b + \sum_{e \in \text{SRC}_v} y_e \right),
\]

where \( \sigma \) is some activation function, \( b \) is a channel-wise bias and the edge transformations \( f_e \) were convolutions with preceding average pooling to provide compatible dimensions for summation.

Most architectures and network designs can be subsumed under this definition of a network, because we do not impose any constraints on the node and edge mappings \( f_v, f_e \). We proceed with a mathematically rigorous definition of network rollouts.

### A1.3 Proofs for definition (rollout) (Section 3)

Let \( N = (V, E) \) be a network. We call a mapping \( R : E \to \{0, 1\} \) a **rollout pattern** of \( N \). For a rollout pattern \( R \), the **rollout window** of size \( W \in \mathbb{N} \) is the directed graph \( R_W = (V_W, E_W) \) with:

\[
V_W := \{0, \ldots, W\} \times V, \quad \forall = (i, v) \in V_W
\]

\[
E_W := \{((i, u), (j, v)) \in V_W \times V_W \mid (u, v) \in E \land j = i + R((u, v))\}. \tag{3}
\]

Edges \( e \in E \) with \( R(e) = 1 \) enable information to directly **stream** through time. In contrast, edges with \( R(e) = 0 \) cause information to be processed within frames before passed through time and, hence, introduce **sequential** dependencies upon nodes inside a frame. We dropped the dependency of especially \( E_W \) on the rollout pattern \( R \) in the notation. A rollout pattern and its rollout windows are called **valid** iff for one and hence for all \( W \in \mathbb{N} \). We denote the set of all valid rollout patterns as \( \mathcal{R}_N \), and the rollout pattern for which \( R \equiv 1 \) the **streaming rollout** \( R^\text{stream} \in \mathcal{R}_N \).

We say two rollout patterns \( R \) and \( R' \) are **equally model-parallel** or **equally parallelizable** iff for all edges \( e = (u, v) \in E \) not originating in the network’s input \( u \notin I_N \) are equal \( R(e) = R'(e) \). For \( i \in \{0, \ldots, W\} \), the subset \( \{i\} \times V \subset V_W \) is called the \( i \)-th **frame**.

**Note (interpretation):** Here, we show how this definition reflects the intuition, that a rollout should be consistent with the network in the sense that it should **contain** all edges / nodes of the network and should not add new edges / nodes, which are not present in the network. Further, we show that this definition yields rollout windows which are temporally consistent and that rollout windows are consistent with regards to each other:

- **Rollout windows cannot add new edges / nodes:** By this, we mean, that a rollout window only contains derived nodes and edges from the original network and for example cannot introduce edges between nodes in the rollout window, which were not already present in the network. This follows directly from the definition of \( E_W \).

- **Edges / nodes of the network are contained in a rollout window:** For vertices this is trivial and for edges \( e = (u, v) \in E \) always \( ((0, u), (R(e), v)) \in E_W \).

- **Rollout windows contain no temporal backward edges:** A backward edge is an edge \( ((i, u), (j, v)) \in E_W \) with \( j < i \). But we know for all edges that \( j = i + R((u, v)) \).

- **Temporal consistency:** Temporal consistency means that for an edge \( ((i, u), (j, v)) \in E_W \) and a second edge between the same nodes \( ((i_*, u), (j_*, v)) \in E_W \) the temporal gap is the same \( j - i = j_* - i_* \). By definition, both are equal to \( R((u, v)) \).

- **Rollout windows are compatible with each other:** We show that \( R_W \) is a sub-graph of \( R_{W+1} \), in the sense that \( V_W \subset V_{W+1} \) and \( E_W \subset E_{W+1} \). From the definition, this is obvious for the set of vertices and edges, but nevertheless we will state if for edges anyway: Let \( \overline{e} \in E_W \) with \( \overline{e} = ((i, u), (j, v)) \). Then by definition \( (u, v) \in E \) and \( j = i + R((u, v)) \). Hence, \( \overline{e} \in R_{W+1} \).
**Proof (well-defined):** For a rollout pattern, we prove that if the rollout window of a certain size $W$ is valid, then the rollout window for any size is valid: Let $R$ be a rollout pattern for a network $N$ and $R_W$ be a valid rollout window. Because $R_W$ hence contains no cycles, also $R_{W'}$ for $W' < W$ contains no cycles (see statement about rollout window compatibility from above). Using induction, it is sufficient to show that $R_{W+1}$ is valid. Assuming it is not, let $p$ be a cycle in $R_{W+1}$. Because there are no temporal backward edges (see above) $p$ has to be contained in the last, the $(W+1)$-th frame. Because of the temporal consistency of rollout windows (see above), there are now cycles in all previous frames which contradicts the validity of $R_W$.

**Proof (streaming rollout exists and is valid):** The streaming rollout pattern $R_{\text{stream}}^\text{stream} \equiv 1$ always exists, because according to our network definition, $E$ is not empty. Further, the streaming rollout pattern is always valid: Assuming that this is not the case, let $R_{W}^\text{stream}$ be a rollout window of size $W$ which is not a-cyclic and let $p$ be a cycle in $R_{W}^\text{stream}$. Because there are no backward edges $\tau = ((i, u), (j, v)) \in E_W$ with $j < i$, all edges of the cycle must be inside a single frame, which is in contradiction to $R_{\text{stream}}^\text{stream} \equiv 1$.

**Note (streaming rollout is un-ambiguous):** Considering the sets of all the most streaming and most-non-streaming rollout patterns

$$R_{\text{stream}} = \left\{ R \in R_N \mid |R^{-1}(1)| = \max_{R_s \in R_N} |R_s^{-1}(1)| \right\}$$

$$R_{\text{non-streaming}} = \left\{ R \in R_N \mid |R^{-1}(0)| = \max_{R_s \in R_N} |R_s^{-1}(0)| \right\}$$

we have shown above that $|R_{\text{stream}}| = 1$ and this is exactly the streaming rollout. In contrast $|R_{\text{non-streaming}}| \geq 1$ especially for networks containing cycles with length greater 1. In this sense, the streaming rollout is un-ambiguous because it always uniquely exists while non-streaming rollouts are ambiguous.

**A1.4 Proof for Lemma (Section 3)**

**Lemma:** Let $N = (V, E)$ be a network. For the number of its valid rollout patterns $|R_N|$, the following holds:

$$1 \leq n \leq |R_N| \leq 2^{|E| - |E_{\text{rec}}|},$$

where $E_{\text{rec}}$ is the set of all self-connecting edges $E_{\text{rec}} = \{(u, v) \in E \mid u = v\}$, and $n$ may be either one of the following:

- $n = 2^{|E_{\text{forward}}|}$, with $E_{\text{forward}}$ being the set of all edges not contained in any cycle of the network.
- $n = \prod_{|C| \in C} (2|C| - 1)$, with $C \subset C_N$ being any set of minimal and pair-wise non-overlapping cycles.

**Proof $|R_N| \leq 2^{|E| - |E_{\text{rec}}|}$:** The number of all (valid and invalid) rollout patterns is $2^{|E|}$, because the pattern can assign 0 or 1 to every edge. In order to be valid (a-cyclic rollout windows), the pattern has to assign 1 at least to every self-connecting edge.

**Proof $1 \leq n$:** Concerning the forward case: According to the definition of a network, $I_N$ is not empty and hence there always exists at least one forward edge $|E_{\text{forward}}| > 0$. Concerning the recurrent case: It is easy to see that $n$ is greater than 0, increases with $|C|$ and that $C$ has to be at least the empty set.

**Proof $n \leq |R_N|$ forward case:** Considering the streaming rollout pattern $R_{\text{stream}}^\text{stream} \equiv 1$ which always exists and is always valid (see above), we combinatorically can construct $2^{|E_{\text{forward}}|}$ different valid rollout patterns on the basis of the streaming rollout pattern by combinatorically changing $R(e)$ for all forward edges $e \in E_{\text{forward}}$. 


Proof \( n \leq |R_N| \) recurrent case: Let \( C \subset C_N \) be any set of minimal and pair-wise non-overlapping cycles. Based on the streaming rollout pattern we will again construct the specified number of rollout patterns. The idea is that every cycle \( p \in C \) gives rise to \( 2^{|p|} - 1 \) different rollout patterns by varying the streaming rollout \( R_{\text{stream}}(E) \equiv 1 \) on all edges in \( p \) and we have to subtract the one rollout for which \( R(p) = 0 \), because for this specific rollout pattern, the cycle \( p \) does not get unrolled. Because the cycle is minimal, those \( 2^{|p|} - 1 \) patterns are different from one another. Because all cycles in \( C \) are disjoint we can combinatorically use this construction across all cycles of \( C \) and constructed \( \prod_{p \in C} (2^{|p|} - 1) \) valid rollouts.

A1.5 Proof update step convergence (Section 3)

Let \( R_W \) be a rollout window for a valid rollout pattern \( R \) of the network \( N = (V, E) \). Then, starting from the initial state and successively applying update steps, converges always to the full state:

\[ \exists n \in \mathbb{N} : U^n(S_{\text{init}}) = S_{\text{full}} \]

Proof: Using induction, we show this without loss of generality for \( R_1 \). Assuming that this is not the case, then there exists a state \( S \in \Sigma_1 \), such that

\[ \forall n \in \mathbb{N} : U^n(S) = S, \text{ and } \exists \sigma = (1, v) \in V_1 : S(\sigma) = 0 \]

But being unable to update \( \sigma \) means, that there is another node that is input to \( \sigma \) which is also not updated yet \((1, v_1) \in V_1 \) and \( S((1, v_1)) = 0 \). Because there are no loops in \( R_1 \) these nodes are not the same \( v \neq v_1 \). This line of argument can now also be applied to \( v_1 \), leading to a third node \((1, v_2) \) with \( S((1, v_2)) = 0 \) and \( v \neq v_1 \neq v_2 \) and so on. Because we only consider networks with \( |V| < \infty \) this leads to a contradiction.

A1.6 Proof Definition of inference update tableau (Section 3)

For a valid rollout pattern \( R \) and a rollout window \( R_W \), we defined the inference update tableau as the following mapping:

\[ T : V_W \rightarrow \mathbb{N} \]

\[ T(\sigma) := \max_{p \in P_{\sigma}} |p| = \arg\min_{n \in \mathbb{N}} \{U^n(S_{\text{init}})(\sigma) = 1\} \]

For this, we have to show, that the equation holds.

Proof: We denote:

\[ T^{\max}(\sigma) := \max_{p \in P_{\sigma}} |p| \]

\[ T^{\min}(\sigma) := \arg\min_{n \in \mathbb{N}} \{U^n(S_{\text{init}})(\sigma) = 1\} \]

and have to show \( T^{\min} = T^{\max} \). The proof is divided into two parts, first showing that the number of necessary inference update steps to update a certain node \( \sigma \) is higher or equal the length of each path \( p \in P_{\sigma} \) and hence \( T^{\min} \geq T^{\max} \). In the second part of the proof we show that maximal paths \( p \in P_{\sigma} \) get successively updated at every update step.

In the first part, we will prove the following statement: For every \( \sigma \in V_W \) and \( p \in P_{\sigma} \):

\[ T^{\min}(\sigma) \geq T^{\min}(p(1)_{\text{src}}) + |p|. \tag{4} \]

Here we denoted again the edges of the path as \( p(i) = (p(i)_{\text{src}}, p(i)_{\text{tgt}}) \in E_W \). In words this means, that for every path in a valid rollout window, the tableau values of the paths first \( p(1)_{\text{src}} \) and last \( \sigma = p(|p|)_{\text{tgt}} \) node differ at least about the length of the path. This is clear for paths of length one \(|p| = 1 \), because \( p(1)_{\text{tgt}} \) can neither be updated before nor at the same update step as \( p(1)_{\text{src}} \), because \( p(1)_{\text{src}} \) is an input of \( p(1)_{\text{tgt}} \). Using induction and the same argument for paths of greater lengths \(|p| = n \) proves (4) and therefore also \( T^{\min} \geq T^{\max} \).

In the second part of the proof, we will show that for all \( \sigma \in V_W \) all paths \( p \in P_{\sigma} \) of maximal length get updated node by node in each update step:

\[ U^{i-1}(S_{\text{init}})(p(i)_{\text{tgt}}) = 0 \]
We will prove this via induction over maximal path lengths. For \( \tau \in V_W \) for which the maximum length of a path \( p \in P_\tau \) is zero \( |p| = 0 \) and hence \( P_\tau = \emptyset \) we know by definition of \( P_\tau \) and because the rollout window is connected to the initial state (see Section A1.5) that \( U^n(S_{\text{init}}(\tau)) = S_{\text{init}}(\tau) = 1 \). This proves the second part for \( \tau \) with maximum path lengths zero. Now we consider \( \tau \in V_W \) for which the maximum length of a path \( p \in P_\tau \) is one \( |p| = 1 \). Because \( p \) is maximal, its first node is in the initial state \( S_{\text{init}}(p(1)_{\text{src}}) = 1 \) and due to the definition of \( P_\tau \) it is \( S_{\text{init}}(p(1)_{\text{tgt}}) = 0 \). Further, because \( p \) is maximal and of length 1, the initial state of all inputs to \( p(1)_{\text{tgt}} \) is 1 and hence \( p(1)_{\text{tgt}} \) can be updated in the first update step \( U(S_{\text{init}}(p(1)_{\text{tgt}})) = 1 \). This proves the second part for \( \tau \) with maximum path lengths one.

Let now be \( n \geq 2 \), and we assume that the statement is true for nodes \( \tau \) for which maximal paths \( p \in P_\tau \) have length \( n \). Be \( \tau \) a node in \( V_W \) for which the maximal length of a path \( p \in P_\tau \) is \( n + 1 \). If the end node of a maximal path \( p \in P_\tau \) cannot be updated \( U^{n+1}(S_{\text{init}}(p(n+1)_{\text{tgt}})) = 0 \), then one of this end node’s inputs \( \tau_{\text{input}} \in V_W \) was not yet updated \( U^n(S_{\text{init}}(\tau_{\text{input}})) = 0 \). But because \( p \) is maximal and of length \( n + 1 \), and \( \tau_{\text{input}} \) is input to \( \tau \), the maximum length of paths in \( P_{\tau_{\text{input}}} \) is \( n \). Hence \( U^n(S_{\text{init}}(\tau_{\text{input}})) = 1 \) contradicting that \( \tau_{\text{input}} \) was not yet updated and therefore proving the second part of the proof. This proves \( T_{\text{min}} = T_{\text{max}} \) and hence both can be used to define the inference update tableau.

### A1.7 Proof for Theorem (Section 3)

**Theorem:** Let \( R \) be a valid rollout pattern for the network \( N = (V, E) \) then the following statements are equivalent:

a) \( R \) and the streaming rollout pattern \( R_{\text{stream}} \) are equally model-parallel.

b) The first frame is updated entirely after the first update step: \( F(R) = 1 \).

c) For \( W \in \mathbb{N} \), the \( i \)-th frame of \( R_W \) is updated at the \( i \)-th update step:

\[
\forall \tau = (i, v) \in V_W : T(\tau) \leq i.
\]

d) For \( W \in \mathbb{N} \), the inference tableau of \( R_W \) is minimal everywhere and over all rollout patterns (most frequent responses & earliest response):

\[
\forall \tau \in V_W : T_{R_W}(\tau) = \min_{R' \in R_N} T_{R'_W}(\tau).
\]

**Proof:** Equivalency of statements a) - d) will be shown via a series of implications connecting all statements:

a) \( \implies \) b): Assuming there is a \( \tau = (1, v) \) which cannot be updated with the first update step, then there has to be an input \( (1, v_{\text{input}}) \) of \( \tau \) for which \( S_{\text{init}}((1, v_{\text{input}})) = 0 \) which contradicts that \( R \) is equally model-parallel to the streaming rollout.

b) \( \implies \) a): Assuming \( R(e) = 0 \) for an edge \( e = (u, v) \in E \) with \( u \notin I_N \), would yield a dependency of \( (1, v) \) on \( (1, u) \). Because \( u \notin I_N \), \( (1, u) \) is not updated at the beginning \( S_{\text{init}}((1, u)) = 0 \) and therefore \( U^1(S_{\text{init}}((1, v))) = 0 \) and hence \( T((1, v)) \geq 2 \) which contradicts b).

c) \( \implies \) b): Trivial.

a) \( \implies \) c): Let \( \tau = (i, v) \in V_W \). First we note, that every maximal path \( p \in P_\tau \) has to start in the initial state \( S_{\text{init}}(p(1)_{\text{src}}) = 1 \), otherwise we can extend \( p \) to a longer path. We will use the definition of \( T \) over maximum path lengths to prove c). Let \( R \) be equally model-parallel to the streaming rollout and \( p \in P_\tau \) a path of maximal length. We know \( S_{\text{init}}(p(1)_{\text{src}}) = 1 \) and hence either \( p(1)_{\text{src}} \in \{0\} \times V \) or \( p(1)_{\text{src}} \in \{0, \ldots, W\} \times I_N \). For the first case, it is easy to see that \( |p| = i \), because \( R \) is equally model-parallel to the streaming rollout and hence one frame is bridged \( R(e) = 1 \) for every edge \( e \) in \( p \). For the second case \( p(1)_{\text{src}} \in \{0, \ldots, W\} \times I_N \), it follows from the same argument as before that \( |p| = i_{\text{src}} \) with \( p(1)_{\text{src}} = (i_{\text{src}}, v_{\text{src}}) \) which proves c).
We define this induced path with $p_R(k) = (j_{src}^k, e_{src}^k, (j_{tgt}^k, e_{tgt}^k)) = ((j_{src}^k, e_{src}^k), (j_{src}^k + R(e^k), e_{tgt}^k))$, for $i \in \{1, \ldots, |p_R|\}$ and $e^k = (e_{src}^k, e_{tgt}^k) \in E$. Let $R'$ be a second valid rollout pattern and let denote $n = |p_R|$. Notice that $j_{tgt}^n = i$. We want to define the induced path $p_{R'} \in P_{\pi}^{R'_W}$ as the path also ending at $\pi \in R'_W$, backwards using the same edges as $p_R$ and respecting the rollout pattern $R'$. We define this induced path $p_{R'} \in P_{\pi}^{R'_W}$ of $p_R$ recursively, beginning with the last edge of $p_R$, as the end of the following sequence of paths, starting with the path:

$$p_{R',1} : \{1\} \rightarrow E_{R'_W}$$

$$p_{R',1}(1) = ((i - R'(e^n), e_{src}^n), (i, e_{tgt}^n))$$

Recursively we define:

$$p_{R',m} : \{1, \ldots, m\} \rightarrow E_{R'_W}$$

$$p_{R',m}(k) = p_{R',m-1}(k-1), \ k \in \{2, \ldots, m\}$$

$$p_{R',m}(1) = ((i - s'_{R',p_R}(m), v_{src}^{n-m+1}), (i - s'_{R',p_R}(m-1), v_{tgt}^{n-m+1}))$$

with $s'_{R',p_R}(m) = \sum_{k=1}^{m} R'(e^{n-k+1})$. In words, $s'_{R',p_R}(m)$ is the frame length of the last $m$ edges of the path $p_R$ under the rollout pattern $R'$. The sequence stops at a certain $m$, either if no edges are left in $p_R$: $m = n$ or at the first time the source of the path’s first edge reaches the 0-th frame: $i - s'_{R',p_R}(m) = 0$. With this definition we can proceed in the prove of a) $\implies$ d):

Let $R$ be equally model-parallel to the streaming rollout pattern, $W \in \mathbb{N}$, and $\pi \in V_W$. Let further be $p_R \in P_{\pi}^{R_W}$ a path of maximal length, $R'$ be any valid rollout pattern, and $p_{R'}$ be the induced path of $p_R$. We want to show that $|p_R| = |p_{R'}|$.

If both rollouts are equally model-parallel on the edges of the path $\{e^1, \ldots, e^{|p_R|}\}$ (this means $R(e^k) = R'(e^k)$ for $k \in \{1, \ldots, |p_R|\}$ if $e^1$ does not originate in the input $e_{src}^1 \notin I_N$, and for $k \in \{2, \ldots, |p_R|\}$ if $e^k$ does originate in the input), the path $p_R$ and its induced path $p_{R'}$ are the same up to their first edge which might or might not bridge a frame, but in both cases $|p_R| = |p_{R'}|$.

If the rollouts are not model-parallel on the edges of the path and hence differ on at least one edge $e^k$ which does not originate in the input, and because $R$ is equally model-parallel to the streaming rollout, it is:

$$s_{R,p_R}(|p_R|) > s_{R',p_R}(|p_{R'}|).$$

Because the induced path using the same rollout cannot loose length, we also know:

$$i - s_{R,p_R}(|p_R|) \geq 0.$$

Greater than zero would be the case for $p_R$ originating in the input $p_R(1)_{src} \in \{1, \ldots, W\} \times I_N$. Combining (5) and (6) yields:

$$i - s_{R',p_R}(|p_{R'}|) > 0.$$ 

Considering the two stopping criteria from the sequence of paths used to define the induced path from above, this proves $|p_R| = |p_{R'}|$.

We now have proven that the induced path $p_{R'}$ from a maximal path $p_R$ in a rollout window from a rollout pattern $R$ which is equally model-parallel to the streaming rollout never is shorter than $p_R$ (especially for highly sequential $R'$, most $p_{R'}$ are not of maximal length). This means, that the maximal length of paths in $P_{\pi}^{R_W}$ is at least as large as the maximal length of paths in $P_{\pi}^{R_W}$ which by definition of the inference update tableau proves a) $\implies$ d).

d) $\implies$ b): Trivial.
Table A1: Experimental setups for the data sets: Images were scaled \((\text{scaling})\); then each frame was perturbed adding Gaussian noise with a standard deviation of \(\sigma\), clipped back to the scaling and for CIFAR10 the channel-wise mean over all training images was subtracted. For GTRSB images of different size were resized \((\text{perturbation})\); data \textit{augmentation} was conducted for training; number of images for training / validation and testing \((\text{training / val. / test size})\) used with a certain \textit{batch size}; experiments were repeated \((\text{reps})\) times.

| Data   | scaling | perturbation | augmentation          | training / val. / test size | batch size | reps |
|--------|---------|--------------|-----------------------|----------------------------|------------|------|
| MNIST  | [0,1]   | 1. \(\mathcal{N}(\sigma = 2.0)\); 2. clipped to [0,1] | None                  | 50k / 10k / 10k           | 128        | 6    |
| CIFAR10| [0,1]   | 1. \(\mathcal{N}(\sigma = 1.0)\); 2. clipped to [0,1]; 3. mean subtracted | horizontal flipping   | 40k / 10k / 10k           | 64         | 1    |
| GTRSB  | [0,1]   | 1. \(\mathcal{N}(\sigma = 0.5)\); 2. clipped to [0,1]; 3. resized to \(32 \times 32\) pixels | None                   | 80% / 10% / 10% of 1305 tracks | 16         | 12   |

Figure A3: (a) A selection of the sequence of networks evaluated on CIFAR10 (for details see Section A2). For evaluating the GTRSB dataset the network DSR4 is used, but without the self-connection of node H1. The input of the networks are images with added Gaussian noise as shown in (b) for CIFAR10 and (c) for GTRSB (for details see Table A1).

A2  Details about networks, data and training

In the depiction of network architectures (Figure 1, Figure A3, and Figure A4) connections between nodes are always realized as convolutional or fully connected layers. In case a node (layer) is the target of several connections, its activation is always computed as the sum of these connection’s outputs. This is mathematically equivalent to concatenating all inputs of the layer and applying a single convolution on the concatenation.

Details about experimental setups and data processing are given in Table A1.

**MNIST**  The network designs are shown in Figure 1 and Figure A4.

The size of the layers (pixels, pixels, features) are: input image \(I\) with \((28, 28, 1)\), hidden layer \(H1\) with \((7, 7, 16)\), hidden layer \(H2\) with \((1, 1, 128)\) and output layer \(O\) with \((1, 1, 10)\).
The following network design specifications were applied with A-B meaning the edge between layer A and layer B. Some of these edges only exist in the network with skip connection (S) or with the skip and self-recurrent connection (SR). For node labels see Figure A4:

- I-H1: a convolution with receptive field 7 and stride 4
- H1-H2 and H2-O: fully connected layers
- H1-O: a fully connected layer
- H1-H1-recurrence: a convolution with receptive field 3 and stride 1

**CIFAR10** The network design is shown in Figure A3. We used a sequence of 7 increasingly deep network architectures with the first network DSR0 being a simple 3 hidden layer forward design and the first hidden layer having a self-recurrent connection. We added additional hidden layers to generate the next networks in the following way: H11 to DSR0, H21 to DSR2, H12 to DSR3, ..., H23 to DSR6.

Note, that every network is a sub-network of its successor. Hence the length of the shortest path is always 4, while the length of the longest path increases from 4 to 11 by 1 for every next network.

The size of the layers (pixels, pixels, features) are: input image I with (32, 32, 3) and output layer O with (1, 1, 10).

The following network design specifications were applied:

- I-H1: a convolution with receptive field 5 and stride 1
- H1-H11, H11-H12, H12-H13: a convolution with receptive field 3 and stride 1
- H2-H21, H21-H22, H22-H23: a convolution with receptive field 3 and stride 1
- H13-H2: a convolution with receptive field 3 and stride 2
- H23-H3: a convolution with receptive field 3 and stride 4
- H3-HD: a fully connected layer
- H1-H1-recurrence: a convolution with receptive field 3 and stride 1
- skip connections H1-H12, H1-H13, H1-H2, H11-H13, H11-H3, H12-H2 and H2-H22, H2-H23, H2-HD, H21-H23, H21-HD, H22-HD: convolution with receptive field 3 and stride

**GTRSB** For the experiments the network DSR4 shown in Figure A3 was used without the H1-H1-recurrence connection. Design specifications have been as described for the CIFAR10 networks with input image I with (32, 32, 3) and output layer O with (1, 1, 10).
For each repetition 80% of the data was randomly taken for training, 10% each for validation and testing.

**Training details** To train networks, we used RMSprop ([51]) with an initial learning rate of $10^{-4}$ and an exponential decay of $10^{-6}$. All networks were trained for 100 epochs. A dropout rate of 0.25 was used for all but the last hidden layer, for which a rate of 0.5 was used. The loss for the rolled-out networks is always the mean over the single-frame prediction losses, for which we used cross-entropy. At the zero-th frame, states of all but the input layers were initialized with zero.
A3 Toolbox for streaming rollouts

One of the contributions of this work is to provide an open source toolbox (code available after review) to design, train, evaluate, and interact with deep networks with a streaming rollout. An example screenshot of the provided graphical user interface is shown in Figure A5.

Networks are specified in a text file, and a core process distributes the network elements onto separate processes and/or GPUs. Elements are executed with alternating read and write phases, synchronized via a core process, and operate on a shared representation of the network. The toolbox is written in Python and uses the Theano [52] or TensorFlow [53] backend. The shared representation enables parallelization of operations across multiple processes and GPUs on one machine and enables online interaction.

Among others, the toolbox provides some of the most widely used optimizers, such as stochastic gradient decent and ADAM [54] and can easily be extended with new ones. Additionally, three types of update estimators are provided to specify plasticities: loss based update, Hebbian based update and parameter regularization update.