LONG SHORT-TERM MEMORY AND LEARNING-TO-LEARN IN NETWORKS OF SPIKING NEURONS

Guillaume Bellec*, Darjan Salaj*, Anand Subramoney*, Robert Legenstein & Wolfgang Maass
Institute for Theoretical Computer Science
Graz University of Technology
Austria
{bellec,salaj,subramoney,legenstein,maass}@igi.tugraz.at
* = first authors

ABSTRACT

Networks of spiking neurons (SNNs) are frequently studied as models for networks of neurons in the brain, but also as paradigm for novel energy efficient computing hardware. In principle they are especially suitable for computations in the temporal domain, such as speech processing, because their computations are carried out via events in time and space. But so far they have been lacking the capability to preserve information for longer time spans during a computation, until it is updated or needed - like a register of a digital computer. This function is provided to artificial neural networks through Long Short-Term Memory (LSTM) units. We show here that SNNs attain similar capabilities if one includes adapting neurons in the network. Adaptation denotes an increase of the firing threshold of a neuron after preceding firing. A substantial fraction of neurons in the neocortex of rodents and humans has been found to be adapting. It turns out that if adapting neurons are integrated in a suitable manner into the architecture of SNNs, the performance of these enhanced SNNs, which we call LSNNs, for computation in the temporal domain approaches that of artificial neural networks with LSTM-units. In addition, the computing and learning capabilities of LSNNs can be substantially enhanced through learning-to-learn (L2L) methods from machine learning, that have so far been applied primarily to LSTM networks and apparently never to SNNs.

This preliminary report on arXiv will be replaced by a more detailed version in about a month.

1 INTRODUCTION

Deep learning in artificial neural networks (ANNs) usually employs feedforward networks. Feedforward networks tend to support computing and learning mechanisms that differ strongly from those of recurrently connected networks of neurons, which one finds in the brain. Recurrent ANNs have been applied very successfully for demanding tasks such as speech processing and video recognition, which require integration of information over time. But virtually all of these applications involve recurrent ANNs that employ specialized modules for preserving information over time, such as Long Short-Term memory (LSTM) units (Hochreiter & Schmidhuber, 1997). A simpler type of memory module had been introduced for artificial neural networks by Mikolov et al. (2014). Modules of this type have been missing in spiking neural network (SNN) models, and as a result, their performance for tasks that require integration over longer timespans has remained inferior. A first step for alleviating this problem was made through the inclusion of short-term synaptic plasticity in randomly connected SNNs (Maass et al., 2002). This improved their capabilities for tasks that require integration of information over hundreds of ms, but did not solve the problem for tasks that involve the behaviourally relevant time scale of seconds and longer.

Neurons in the brain are subject to intracellular processes on many different times scales, from ms to s, minutes, hours, and days. We show that by including just one of their longer lasting processes, neuronal adaptation, and by simultaneously considering structured rather than randomly connected
network architectures, one arrives at recurrent SNNs with substantially enhanced computing and learning capabilities. We will refer to these SNNs with longer short-term memory as LSNNs.

2 THE LSNN MODEL

We use simple leaky integrate-and-fire neuron models (see Supplementary Information). One population R of regularly firing neurons in the LSNN consists of standard neurons of this type, whereas a simple adaptation mechanism is included for neurons in a second subpopulation A. Both populations receive spike trains from a population X of external input neurons. Results of computations in the LSNN are read out by a population Y of external linear readout neurons, see Fig. 1A.

In neuroscience the term neural adaptation refers to a broad range of phenomena and mechanisms. It was already discovered in the 19th century that the intensity of a constant stimulus tends to cause a diminishing sensation, see e.g. the Wikipedia article “Neural adaptation”. A closer look shows that these phenomena take place simultaneously on several time scales, from hundreds of ms to s, hours, or even days. In the context of neuron models, adaptation refers to changes in the firing threshold or subthreshold currents of a neuron that reduce its excitability in response to its own firing (Gerstner et al., 2014). Experimental data on that are documented for example in the form of a distribution of measured adaptation indices of neurons for large sets of neurons, both from mouse and humans, in the online Allen Brain Atlas (Allen Institute, 2018). This adaptation index measures the increase of interspike intervals of a neuron that receives a constant input current. Common ways for modeling such data are described in (Gerstner et al., 2014; Pozzorini et al., 2015; Gouwens et al., 2018; Teeter et al., 2018). We are using here the arguably simplest form of integrating adaptation into a neuron model: We assume that the firing threshold $B_j(t)$ of neuron $j$ increases by some fixed amount $\beta$ for each spike of this neuron $j$, and then decays exponentially back to a baseline value $b_0^j$ with a time constant $\tau_{a,j}$. The neural dynamics for a discrete time step of $\delta t = 1$ reads as follows

\begin{align}
B_j(t) &= b_0^j + \beta b_j(t), \\
b_j(t+1) &= \rho_j b_j(t) + (1 - \rho_j) z_j(t),
\end{align}

where $\rho_j = \exp(-\frac{\delta t}{\tau_{a,j}})$ and $z_j(t)$ is the spike train of neuron $j$. Note that this dynamics of thresholds of adaptive spiking neurons is similar to the dynamics of the state of context neuron in (Mikolov et al., 2014).

We optimize the weights, and in some cases also the connections of an LSNN through standard optimization algorithms for specific ranges of tasks. This optimization algorithm is not claimed to be biologically realistic. But it might correspond functionally to evolutionary processes that optimized LSNNs in the brain for tasks that were relevant for survival. For the three experiments that we report here we used backprop through time (BPTT). BP has already been applied by (Courbariaux et al., 2016) and (Esser et al., 2016) to feedforward networks of spiking neurons, and by (Hub & Sejnowski, 2017) to recurrent networks of spiking neurons. The gradient is backpropagated through spikes by replacing in these approaches the non-existent derivative of the membrane potential at the time of a spike by a pseudo-derivative that smoothly increases from 0 to 1, and then decays back to 0. For our applications we needed to modify the pseudo-derivative in order to achieve stable performance of BPTT for larger time spans, that required backpropagation through several $1000\times$ layers of an unrolled feedforward network of spiking neurons: we reduced (“dampened”) the amplitude of the pseudo-derivative by a factor $< 1$.

3 LONG SHORT-TERM MEMORY IN LSNNs

Store-Recall task:

We first test the performance of LSNNs on a very simple working memory task (see Fig. 1D): The network has to store a bit – encoded by spikes from input neurons – in working memory if it receives simultaneously a “store” instruction, which is given in the form of spikes from other input neurons. The stored bit has to be recalled by the network after a subsequent “recall” instruction, that is also given in the form of spikes from dedicated input neurons. At the same time other spike inputs should be ignored by the network.
Figure 1: **Store-recall task.** A An LSNN consists of a regular spiking and an adaptive neural population. B A standard LSNN architecture that corresponds to the ANN architecture of Mikolov et al. (2014). C A sample LSNN architecture that resulted from using Deep R, a biologically inspired technique that prunes and rewires gradually the connectivity matrices during training (Bellec et al., 2018). D The raster plot shows the activity of a network after convergence. From top to bottom, the sub-panels show spike rasters from input neurons (X), regular spiking neurons (R), and adaptive neurons (A) followed by the output of the readout and the dynamics of the adaptive thresholds of adaptive neurons (this network is based on the standard architecture).
More precisely, the input neurons marked in blue and red at the top of Fig. 1D are Poisson neurons that produce random spike sequences. An input bit to the network is encoded by spiking activity at 50 Hz in the corresponding input channels $X_0$ or $X_1$ for a pattern duration $D = 200$ ms. “Store” and “recall” instructions are analogously encoded through firing of populations $X_S$ and $X_R$ of other Poisson input neurons at 50 Hz for $D = 200$ ms. Otherwise input neurons are assumed to be silent in this simple experiment.

When a “recall” instruction is received, the network should output the bit value received concurrently with the most recent “store” instruction before that. After a “store,” a “recall” instruction is given during a subsequent time period of length $D$ with probability $\pi$. This results in an expected delay of $D/\pi$ between a “store” and “recall” instruction. The next “store” appears similarly in a subsequent period of length $D$ with probability $\pi$.

During a recall cue, the binary output is supposed to be encoded by the sign of the time-averaged membrane potential of a linear readout neuron $Y$. The LSNN consisted of 60 regular spiking and 20 adaptive neurons. An example trial after training is shown in Fig. 1D.

A summary of results on performance of trained networks for several versions of the store-recall tasks is given in Tables 1 and 2 of the Supplement. A SNN without adapting neurons can solve the store-recall task only if working memory for a few 100 ms suffices. LSNNs work best if the time constants of their adapting neurons are in the same range as the required durations of working memory in the network. If the adapting neurons of an LSNN have a spread of different time constants, such LSNN generally performs well for a variety of tasks that require working memory durations with a similar spread of memory durations. Table 2 shows that the architecture of an LSNN also influences its performance for these tasks. In general, the standard architecture (Fig. 1B) works quite well. But first tests where the network architecture was optimized together with its parameters suggest that more specific architectures, such as the one shown in Fig. 1C may perform substantially better for specific tasks (see lowest row of Table 2). We applied there the rewiring method Deep R (Bellec et al., 2018). Deep R converges theoretically to an optimal network configuration by continuously updating the set of active connections in a biologically inspired manner (Kappel et al., 2015, 2017; Bellec et al., 2018).

**Sequential MNIST:** We next tested the performance of LSNNs on a task that requires continuous updates of working memory over a longer time span: sequential MNIST. The task is to classify the handwritten digits of the MNIST datasets when the pixels are presented one after another.

We compare the performance of LSNNs with that of LSTM networks. For fair comparison, the size of the LSNN is chosen to match the number of parameters of the LSTM network. This leads to 140 regular spiking and 100 adaptive neurons with adaptation time constant 700 ms in the LSNN, in comparison to 128 LSTM units.

The pixels are presented sequentially in 784 time steps, after which the network is required to output the class of the presented digit. For LSNNs, the gray level of each pixel is encoded into the spikes of 80 input neurons. Several different ways of encoding pixel values by spikes can be used. Fig. 2C shows a version where the gray level of each pixel is encoded by population coding through the firing probability of the 80 input neurons, 1 ms for each pixel. Somewhat better performance is achieved when each of the 80 input neurons is associated with a particular threshold for the gray value, and this input neuron fires whenever the gray value of the currently presented pixel crosses its threshold (two input neurons were used per threshold, one spiked at threshold crossings from below, and one at the crossings from above; this input convention is chosen for the LSNN results of Fig. 2B). In either case, an additional input neuron becomes active when the presentation of the 784 pixel values is finished, in order to prompt an output from the network. The firing of this additional input neuron is shown at the top right of the top panel of Fig. 2C. The softmax of 10 linear output neurons $Y$ is trained to produce, during that time segment, the label of the sequentially presented handwritten digit (see yellow shading around 800 ms of the output neuron for label 6 in the plot of the dynamics of the output neurons $Y$ in Fig. 2C: this output was correct). In contrast to the store-recall task, we see that the firing thresholds of adapting neurons undergo complex temporal modifications.

The output of the network is determined by averaging the readout output over the 56 ms following the presentation of the digit. The network is trained by minimizing the cross entropy between the softmax of the averaged readout and the label distributions.
Figure 2: **Sequential MNIST.** A The goal of this task is to classify images of handwritten digits when the pixels are shown sequentially row by row. B The performance of spiking neural networks is compared for three different setups: without adapting neurons (LIF), an LSNN in the standard architecture, but with connections from the regular to the adaptive spiking pool (LSNN), and an LSNN with randomly initialized connectivity that was rewired during training (LSNN rewired). For comparison, the performance of an artificial fully connected RNN and an LSTM with standard gray level inputs are shown. C Dynamics of a network classifying a digit as in Figure 1.
A performance comparison is given in Figure 2B. LSNNs achieve a 93% classification accuracy on the test set using the standard architecture with added connections from the regular to the adaptive spiking pool. A higher performance of 96% is reached if the network is rewired during training with DEEP R (Bellec et al., 2018). The same performance is achieved by an LSTM receiving continuous gray levels instead of spikes. The first and fourth bars in Fig. 2B show that this range of accuracy is out of reach for spiking or nonspiking neural networks without enhanced working memory capabilities.

Each image presentation with this setup takes 784 ms. To test if the model can also integrate information over a longer time span we expanded this duration to 3.36 s by presenting each pixel for 4 milliseconds instead of 1. An LSNN trained on this expanded presentation duration achieved in a first experiment 83% classification accuracy (all adaptive neurons had an adaptation time constant of 3500 ms).

4 LEARNING-TO-LEARN FOR LSNNs

Some of the most intriguing applications of LSTM networks were demonstrations of Learning-to-Learn (L2L) or Meta-learning capabilities of this type of neural networks. This research direction was initiated by Hochreiter et al. (2001), and has lead in recent years to a host of remarkable new methods and results, see e.g. Wang et al. (2016) and Gupta et al. (2018). The reason why LSTM networks have been used for most work on L2L is that they can naturally accommodate two levels of learning and representation of learned insight: network parameters such as synaptic weights can be used to encode “innate” or transfer knowledge from preceding learning of more-or-less related tasks. At the same time, a sufficiently long short-term memory in the neural network can accumulate and apply knowledge gained during a single learning iteration (generally consisting of many episodes) for a specific learning task. Of course, many other schemes for encoding, fusing, and applying results of long-term meta-learning and results from the current learning task are possible and should be explored.

The results of Hochreiter et al. (2001) show that besides backprop there are likely to be many other methods which recurrent neural networks can use to learn nonlinear functions from a teacher. This supervised learning scenario is also of interest from the perspective of understanding learning in the brain, since learning to predict future sensory inputs, the consequences of an action, or sensory input from one modality in terms of sensory inputs from other modalities, are all learning scenarios where the target output values are given to networks of neurons in the brain – with often some delay – from the environment or internal processing modules. Another aspect of this L2L approach is that it produces a host of new ideas and algorithms for implementing learning in recurrent neural networks. This is of interest from the perspective of neuroscience, because on the basis of the currently available experimental data it is not clear to what extent synaptic weights are able to support fast learning by storing knowledge from previous trials for the current learning task. It is also not clear whether spike-timing-dependent plasticity (STDP) is the primary local learning rule in neural networks of the brain, and which other local learning rules are there implemented (Lisman & Spruston, 2010; Titley et al., 2017). In addition, experimental data for STDP question its usefulness for fast and reliable learning, since STDP is subject to high synapse-to-synapse variability, and requires dozens of repetitions of pairing of pre- and postsynaptic firing (Froemke et al., 2010).

We show here that the L2L paradigm of Hochreiter et al. (2001) for LSTM networks can be ported to the biologically more realistic LSNNs. L2L results for SNNs have the advantage that many features and fingerprints of resulting new learning algorithms for SNNs, such as neural activity (spike trains), changes in the excitability of neurons and synaptic weights, and changes in the network dynamics through learning can be compared with corresponding measurements from networks of neurons in the brain. At the same time, the resulting new methods for learning nonlinear functions are also of interest for spike-based neuromorphic chips such as Brainscaler (Schemmel et al., 2010), SpiNNaker (Furber et al., 2013), True North (Esser et al., 2016), chips from ETH Zurich such as (Qiao et al., 2015), and Loihi (Davies et al., 2018). Nonlocal learning rules such as backprop are real challenges for these neuromorphic devices, and alternative local rules for learning nonlinear functions are therefore of particular interest.

The standard setup of L2L involves a large, possibly infinitely large, family $F$ of learning tasks $C$. Some parameters $P$ of a neural network $N$, in our case an LSNN, are optimized in an outer
optimization loop to support fast learning of a randomly drawn task \( C \) from \( \mathcal{F} \). Like in (Hochreiter et al., 2001) we let all synaptic weights \( \mathbf{W} \) of \( \mathcal{N} \) belong to the set of parameters \( \mathbf{P} \) that are optimized through the outer loop, i.e., through meta-learning. Hence the network is forced to encode all results from learning the current task \( C \) in its internal state, in particular in its short-term memory. Thus the synaptic weights \( \mathbf{W} \) of the neural network \( \mathcal{N} \) encode an efficient algorithm for learning arbitrary tasks \( C \) from \( \mathcal{F} \), rather than results of learning a particular task \( C \). The teacher input for learning a particular function \( C \) from \( \mathcal{F} \) is given in (Hochreiter et al., 2001) in a delayed manner: The target output value of \( C \) is given after \( \mathcal{N} \) has provided its guessed output value for the preceding input values. This avoids that \( \mathcal{N} \) learns to cheat by simply using the teacher input as its guessed output value. We are using the same convention in our experiment.

The learning methods that can possibly be encoded by the synaptic weights \( \mathbf{W} \) range over a substantially larger range of local update rules than commonly considered for network learning. Note for example that most commonly considered local update rules are formulated as functions where a parameter update (e.g., a weight update) depends in a particular but smooth manner on previous parameter values and specific aspects of the recent network activity, such as spike times or rates of particular neurons. As long as this function is smooth, it can be approximated by a neural network. But importantly, ANY smooth update function – even a function that was previously never considered for neural network learning can be approximated by a neural network. The L2L approach of (Hochreiter et al., 2001) allows in principle in the outer optimization loop a search over the best performing ones among all these potential local update rules. They are only restricted to those that can be implemented by the neural network \( \mathcal{N} \). In particular, they are constrained to local learning rules if the neural network \( \mathcal{N} \) only supports local rules for computing and changing the content of short-term memories, as is the case for the LSNNs \( \mathcal{N} \) that we are considering here. For example, the network \( \mathcal{N} \) is free to use the delayed teacher input for computing the error that it made on the preceding step, and to transmit this error through its internal spiking activity to other neurons in \( \mathcal{N} \), in particular also to adapting neurons.

One of the more challenging family \( \mathcal{F} \) of tasks considered by (Hochreiter et al., 2001) for LSTM networks was the class of all quadratic functions over two real-valued variables. We found that L2L for this family \( \mathcal{F} \) works also very well for LSNNs. We present here instead a potentially even more interesting variation of this L2L task, where \( \mathcal{F} \) consists of all functions of two real-valued variables that can be realized by a minimal class of artificial neural networks: small two-layer target neural networks TN, i.e., feedforward networks with one hidden layer, consisting of sigmoidal (non-spiking) neurons. We found that LSNNs can learn to learn quickly to reproduce the input-output behavior of any such ANN TN, by just using their network dynamics and their internal states for learning and emulating this ANN. Note that BPTT was only used for the outer learning loop, in order to determine the synaptic weights \( \mathbf{W} \) of the LSNN. But these could not be optimized for a particular target network TN. Constraints of compute time did not yet allow us to emulate learning for larger target networks TN, and we also did not try to optimize the architecture of the LSNN for this task. Hence the results below provide just a proof of concept.

The target network TN represents a function \( C(x_1, x_2) \), with 2 inputs \( x_1 \) and \( x_2 \) and 2 hidden neurons, with a sigmoid activation at both the hidden and output layers (shown in Fig. 3A). The outputs of this function, which serve as the targets for the output \( Y \) of the LSNN, are scaled to be between \([0, 1]\).
Figure 3: (Caption next page.)
The training procedure was as follows: Network training was divided into training episodes. At the start of each training episode, a new set of input pairs \( x_1, x_2 \) and new weights (between \([-1,1]\)) for the target network TN were randomly chosen and used to generate targets \( C(x_1, x_2) \). These input pairs and targets were used as training data and presented to the LSNN during the episode. The LSNN parameters were updated using BPTT to minimize the mean squared error between the LSNN output and the target in the training set, using gradients computed over batches of 10 such episodes. In other words, each weight update included gradients calculated on the input/target pairs from 10 different TNs, which prevented the LSNN from specializing on predicting the output of one single TN.

In each episode, for the given TN, 500 different input/target pairs were presented, one per step. Each step lasted for 20 timesteps, which corresponds to 20 ms. The LSNN received at each step three input values as input: \( x_1, x_2 \) and \( C(x'_1, x'_2) \), where \( x'_1, x'_2 \) were the inputs for the preceding step (i.e., \( C(x'_1, x'_2) \) was the target of the previous step; it was set to 0 in the first step) during both the training and testing process, as illustrated in Fig. 3B. While testing, the weights of the LSNN remained fixed, and it was required to predict targets from functions \( C_{test} \in F \) that it had never seen before.

The input population consisted of three subpopulations of 100 neurons each, one population for each of the inputs \( x_1, x_2 \) and \( C(x'_1, x'_2) \). These input populations produced Poisson spike trains based on a Gaussian population-rate encoding of the input values, between 0 and 200 Hz. The range of each input was split evenly among all the neurons responsible for that input, and for each given analog input value, the firing rate of the neurons had a Gaussian activity profile with mean at that analog value, and a variance of 0.1.

We used an LSNN architecture that contained, in addition to the connections of the standard architecture, connections from the regular-spiking to the adaptive population (see Fig. 3B), since performance of the standard architecture was much worse. The network contained 100 in each of the regular spiking and adaptive populations. A sample spike raster is shown in Fig. 3F (which is for the same episode as shown in Fig. 3D). The top panel shows the input values.

During training, there was an initial steep drop in error (see Fig. 3C) when the network learned to output values in the same range as the target functions, see Fig. 3E left and middle. The performance of the network was compared to a linear baseline: for each TN, an optimal linear regressor was computed using the same input/target pairs presented to the LSNN. The mean squared error for the predictions of this linear regressor on these input/target pairs was the linear baseline.

During testing, even though the network was shown target functions \( C_{test}(x_1, x_2) \) it had not seen before, the test error remained low. The temporal evolution of test errors for the first 100 steps in a sample test episode for a new target function \( C \) is shown in Fig. 3D. It can be seen that the network had a high error in the first few steps of the episode, but it quickly learned to output the correct targets, even though its weights \( W \) remained fixed. Thus the LSNN had learnt to learn in other ways than through changes of its synaptic weights.
5 Discussion

We have shown that the inclusion of adapting neurons significantly enhances computing and learning capabilities of recurrent networks of spiking neurons. In particular we have shown that the resulting LSNN is able to integrate and maintain information over several seconds in a working memory. In fact, it is reasonable to assume that any duration for working memory can be achieved with corresponding large time constants of adapting neurons. Furthermore, our tests so far suggest that LSNNs are able to approach the computational power and learning capability of the generally best performing ANN model for temporal processing tasks: LSTM networks. BPTT tends to work well for LSNNs because the error gradient can be propagated through the slowly changing firing threshold of an adapting neuron without causing vanishing or exploding gradients. Hence adapting neurons appear to provide “highways” for propagating errors backwards even over 100s and 1000s of layers of an unrolled LSNN.

A remarkable feature of a realization of working memory in SNNs through increased firing thresholds is that this method is very energy-efficient: When a memory is stored through an increased firing threshold, this tends to reduce firing activity in the SNN. In contrast, working memory was previously modelled in SNNs via persistent firing of a subset of neurons, i.e., through increased firing activity. In neuroscience there is an ongoing debate to what extent working memory is implemented in the brain through persistent firing or through activity-silent mechanisms (Stokes, 2015). Interestingly, experimental data show that also one form of long-term memory, familiarity, is expressed in some brain regions (perirhinal and entorhinal cortex) through reduced activity during the presentation of a familiar stimulus (Eichenbaum et al., 2007). These results are consistent with a memory model based on longer lasting increases of the firing threshold of specific subsets of neurons, see also (Titley et al., 2017).

The implementation of short-term memory through neural adaptation is somewhat analogous to an interesting variant of long short-term memory in artificial neural networks (Mikolov et al., 2014). They introduced a particular population of neurons, termed context neurons, whose state changes on a slower time scale than that of the other neurons. If one interprets the firing threshold of adapting neurons as their “state”, then the dynamics of this state is analogous to the dynamics of context neurons in their approach. However, in the non-spiking setting of (Mikolov et al., 2014) one cannot see an advantage of this model with regard to energy-efficiency of a physical realization: the state of context neurons has to be read out at every discrete time step, like for all other neurons. In other words, a clocked network cannot make use of the possibility to convey information by not sending a message, as one can do in SNNs. This option, to maintain and use information by not firing, is extensively used by LSNNs.

In (Costa et al., 2017) an LSTM module was modelled through a recurrent network of non-spiking neurons, i.e., of sigmoidal neurons whose analog outputs are interpreted as firing rates. Their model is consistent with models for working memory based on persistent firing. It achieved remarkable results for several datasets, including the same sequential MNIST data that we have considered for the case of spike-based network inputs and spike-based network computations.

We have shown in Fig. 3 that the inclusion of adapting neurons in SNNs opens the door to porting methods and results on L2L from LSTM networks to SNNs. Our result suggests new ways of storing the “program” of a computation in SNNs: In the states of adapting neurons, i.e., in the working memory of the LSNN. Similarly as previously shown for artificial neural networks in (Hochreiter et al., 2001), this approach simultaneously opens the door to the exploration of new classes of learning algorithms for SNNs that could previously not be considered. In particular, our result shows that LSNNs can learn the input/output behaviour of feedforward artificial neural network with a hidden layer by just using readily available local mechanisms. This shows that LSNNs are able to emulate the function of backprop learning in quite different ways than considered so far.

We will show in (Bellec et al., 2018) that LSNNs also pave the way for porting new ideas for learning to learn from rewards (Wang et al., 2016) to networks of spiking neurons. In particular, one can see that LSNNs are able to learn nontrivial sequential behaviors, and that they can make use of control transfer learning, as shown there for LSTM networks.
ACKNOWLEDGMENTS

Research leading to these results has in parts been carried out on the Human Brain Project PCP Pilot Systems at the Juelich Supercomputing Centre, which received co-funding from the European Union (Grant Agreement #604102). We gratefully acknowledge Sandra Díaz, Alexander Payser and Wouter Klijn from the Simulation Laboratory Neuroscience of the Jülich Supercomputing Centre for their support. This research was carried out under partial support by the Human Brain Project of the European Union #720270 and #785907.

REFERENCES


**Supplementary Information**

**Neuron model:** In continuous time the spike trains $x_i(t)$ and $z_j(t)$ are formalized as sums of Dirac pulses. Neurons are modeled according to a standard adaptive leaky integrate-and-fire model. A neuron $j$ spikes as soon as its membrane potential $V_j(t)$ is above its threshold $B_j(t)$. At each spike time $t$, the membrane potential $V_j(t)$ is reset by subtracting the current threshold value $B_j(t)$. Importantly at each spike the threshold $B_j(t)$ of an adaptive neuron is increased by a constant $\beta$. Then the threshold decays back to a baseline value $b^0_j$. Between spikes the membrane voltage $V_j(t)$ and the threshold $B_j(t)$ are following the dynamics

$$\tau_m \dot{V}_j(t) = -V_j(t) + R_m I_j(t)$$
$$\tau_{a,j} \dot{B}_j(t) = b^0_j - B_j(t),$$

where $\tau_m$ is the membrane time constant, $\tau_{a,j}$ is the adaptation time constant and $R_m$ is the membrane resistance. The input current $I_j(t)$ is defined as the weighted sum of spikes from external inputs and other neurons in the network:

$$I_j(t) = \sum_i W_{ji}^{\text{in}} x_i(t - d_{ji}^{\text{in}}) + \sum_i W_{ji}^{\text{rec}} z_i(t - d_{ji}^{\text{rec}}),$$

where $W_{ji}^{\text{in}}$ and $W_{ji}^{\text{rec}}$ denote respectively the input and the recurrent synaptic weights and $d_{ji}^{\text{in}}$ and $d_{ji}^{\text{rec}}$ the corresponding synaptic delays. All network neurons are connected to a population of readout neurons with weights $W_{kj}^\text{out}$. When network neuron $j$ spikes, the output synaptic strength $W_{kj}^\text{out}$ is added to the membrane voltage $y_k(t)$ of all readout neurons $k$. $y_k(t)$ also follows the dynamics of a leaky integrator $\tau_m y_k(t) = -y_k(t)$. The physical units of voltages, currents, delays, Dirac pulses and synaptic weights are respectively Volt, Ampere, Second, Hertz and Coulomb.

**Implementation in discrete time:** Our simulations were performed in discrete time with a time step $\delta t = 1$ ms. In discrete time, the spike trains are modeled as binary sequences $x_i(t), z_j(t) \in \{0, 1\}$. Neuron $j$ emits a spike at time $t$ if it is currently not in a refractory period, and its membrane potential $V_j(t)$ is above its threshold $B_j(t)$. During the refractory period following a spike, $z_j(t)$ is fixed to 0. The dynamics of the threshold is defined by $B_j(t) = b^0_j + \beta_j(t)$ where $\beta$ is a constant which scales the deviation $b_j(t)$ from the baseline $b^0_j$. The neural dynamics in discrete time reads as follows

$$V_j(t + 1) = \alpha V_j(t) + (1 - \alpha) R_m I_j(t) - B_j(t) z_j(t)$$
$$b_j(t + 1) = \rho_j b(t) + (1 - \rho_j) z_j(t),$$

where $\alpha = \exp(-\frac{\delta t}{\tau_m})$ and $\rho_j = \exp(-\frac{\delta t}{\tau_a})$. The term $B_j(t) z_j(t)$ implements the reset of the membrane voltage after each spike. The current $I_j(t)$ is the weighted sum of the incoming spikes. The definition of the input current in equation [5] holds also for discrete time, with the difference that spike trains now assume values in $\{0, 1\}$.

**Parameter values:** For adaptive neurons, we used $\beta_j = 1.7$, and for regular spiking neurons we used $\beta_j = 0$ (i.e. $B_j$ is constant). The baseline threshold voltage was $b^0_j = 0.01$ and the membrane time constant $\tau_m = 20$ ms. Initial network weights were drawn from a Gaussian distribution $W_{ji} \sim \mathcal{N}(0, 1)$, where $n_{in}$ is the number of afferent neurons in the considered weight matrix (i.e., the number of columns of the matrix), $\mathcal{N}(0, 1)$ is the zero-mean unit-variance Gaussian distribution, and $w_0$ is a weight scaling factor chosen to be $w_0 = \frac{1V}{R_m \delta t}$. For this choice, a spike transmitted with synaptic weight $w_0$ resulted in an increase of post-synaptic potential of $1 - \alpha$. In this way, the standard initialization of synaptic weights initialized the networks in a stable regime producing realistic firing rates.

In store-recall the test performance is computed over sequences of 32 consecutive store-recall pairs and averaged over 32 random input sequences. During training with BPTT the depth of the unrolled network is adapted to the distribution of store-recall delays. For expected delays of 2, 4, and 6 s it resulted in 3200, 5200 and 8000 unrolled time steps respectively. All networks were trained for 500 iterations. We used the Adam optimizer \cite{Kingma&Ba2014} with default parameters and a
learning rate of 0.01, with sequences presented in batches of sizes indicated in Table 1 and 2 (as "batch size"). To avoid unrealistically high firing rates, the loss function contains a regularization term that minimizes the distance of average firing rate between individual neurons and a target firing rate of 10 Hz.

In sequential MNIST all networks were trained for 36,000 iterations with approximately 66,000 parameters, with a batch size of 256, using the default Adam optimizer, and a learning rate initialized at 0.01 and decayed by a factor 0.8 every 2500 iterations. The baseline artificial RNN contains 128 hidden units and uses hyperbolic tangent as the activation function. The LIF network is formed by a fully connected population of 220 regular spiking neurons.

In the learning to learn task, the network was trained for 5,000 iterations with a batch size of 10 episodes, using the RMSProp ([Hinton et al., 2012] optimizer, and a learning rate of 0.001.

**Propagation of gradients in recurrent networks of LIF neurons:** In artificial recurrent neural networks such as LSTMs, gradients can be computed with back propagation through time (BPTT). For BPTT in spiking neural networks, complications arise from the non-differentiability of the output of spiking neurons, and from the fact that gradients need to be propagated either through continuous time or through many time steps if time is discretized. Therefore, in [Courbariaux et al., 2016; Esser et al., 2016] it was proposed to use a pseudo-derivative.

\[
\frac{dz_j(t)}{dv_j(t)} := \max\{0, 1 - |v_j(t)|\},
\]

where \(v_j(t)\) denotes the normalized membrane potential \(v_j(t) = \frac{V_j(t) - B_j(t)}{B_j(t)}\). This made it possible to train deep feed-forward networks of deterministic binary neurons ([Courbariaux et al., 2016; Esser et al., 2016]). We observed that this convention tends to be unstable for very deep (unrolled) recurrent networks of spiking neurons (see Fig. 1E). To achieve stable performance we dampened the increase of back propagated errors through spikes by using a pseudo-derivative of amplitude \(\gamma < 1\) (typically \(\gamma = 0.3\)):

\[
\frac{dz_j(t)}{dv_j(t)} := \gamma \max\{0, 1 - |v_j(t)|\}.
\]

Note that in adaptive neurons, gradients can propagate through many time steps in the dynamic threshold. This propagation is not affected by the dampening.
Table 1: Test error (%) on store-recall tasks with different delays, and different time constants of adapting neurons in LSNNs with the standard architecture. Mean test errors (over 3 random network initializations on test batch of size 512) reached by different networks after 500 training iterations. Best and second best results per task (column) are marked with boldface ($D$: pattern presentation time). For comparison, the performance of a fully connected recurrent SNN without adapting neurons is shown in the last row (LIF).

<table>
<thead>
<tr>
<th>expected delay between “store” and “recall” instructions</th>
<th>200ms</th>
<th>2s</th>
<th>4s</th>
<th>6s</th>
<th>uniform in [0-8]s</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$ (ms)</td>
<td>50</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>batch size</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>256</td>
</tr>
<tr>
<td>LSNN $\tau_a = 200$ms</td>
<td>0.11</td>
<td>51</td>
<td>48</td>
<td>48</td>
<td>49</td>
</tr>
<tr>
<td>LSNN $\tau_a = 2s$</td>
<td>0.24</td>
<td>1.59</td>
<td>1.97</td>
<td>15.7</td>
<td>16.9</td>
</tr>
<tr>
<td>LSNN $\tau_a = 4s$</td>
<td>0.30</td>
<td>7.62</td>
<td>3.77</td>
<td>8.27</td>
<td>6.96</td>
</tr>
<tr>
<td>LSNN $\tau_a = 6s$</td>
<td>0.58</td>
<td>7.24</td>
<td>9.17</td>
<td>5.06</td>
<td>3.44</td>
</tr>
<tr>
<td>LSNN $\tau_a = 8s$</td>
<td>0.40</td>
<td>16.3</td>
<td>45.1</td>
<td>34.0</td>
<td>8.33</td>
</tr>
<tr>
<td>LSNN $\tau_a$ spread over [0, 8]s</td>
<td>0.41</td>
<td>3.08</td>
<td>3.25</td>
<td>8.12</td>
<td>4.36</td>
</tr>
<tr>
<td>LIF</td>
<td>0.08</td>
<td>49</td>
<td>49</td>
<td>51</td>
<td>49</td>
</tr>
</tbody>
</table>

Table 2: Test error (%) of different LSNN architectures on two store-recall tasks. Mean test errors (over 3 random network initializations on test batch of size 512) reached by different networks after 500 training iterations. Rows show results for SSNs with different architectures. The adaptation time constants were spread for all these networks uniformly over [0, 8] s, like in the second to last row of Table 1.

<table>
<thead>
<tr>
<th>expected delay</th>
<th>4s</th>
<th>uniform in [0-8]s</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$ (ms)</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>batch size</td>
<td>24</td>
<td>256</td>
</tr>
<tr>
<td>Standard</td>
<td>3.25</td>
<td><strong>4.36</strong></td>
</tr>
<tr>
<td>Fully connected</td>
<td>6.39</td>
<td>31.8</td>
</tr>
<tr>
<td>Deep R</td>
<td><strong>0.41</strong></td>
<td>5.17</td>
</tr>
</tbody>
</table>