Biologically plausible learning and meta-learning in recurrent networks of spiking neurons

by

Anand Subramoney

DISSERTATION

submitted for the degree of

Doctor Technicae

Institute for Theoretical Computer Science

Graz University of Technology

Thesis Advisor

Prof. Dr. Wolfgang Maass

Graz, May 2020

This document is set in Palatino, compiled with pdfLaTeX2e and `Biber`.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

The wide repertoire of abilities of the human brain poses an open challenge to understand how these abilities could be reproduced computationally, on the path to understanding intelligence and the brain. These abilities include various cognitive tasks, learning the structure of the environment and using it to adapt quickly, and learning to perform complex motor tasks within the limitations of what is possible in biology. In biology, plasticity and learning are constrained by the architecture and processes present in the brain. An understanding of the inductive biases and learning rules required for solving such tasks, and a computational analysis of emergent representations when performing such tasks has so far been missing. Moreover, while fast adaptation is a well known phenomena, most of the current theories revolve around studying it in the context of plasticity. This is in spite of the fact that there are few plasticity processes in biology that are known to involve fast enough processes to explain this rapid adaptation. To address these issues, an inductive bias that makes it possible for recurrent networks of spiking neurons (RSNNs) to solve tasks involving flexible cognitive control and computation — spike frequency adaptation — is studied. Then the installation of additional inductive biases in RSNNs using optimization rather than manual design is illustrated. It is shown that RSNNs can learn to learn/adapt quickly, where the adaptation can happen either using just the plasticity of a readout, or no synaptic plasticity at all. Adaptation without synaptic plasticity can occur on very short time scales, since it is not limited by the time scales of plasticity — the adaptation occurs solely using the dynamics of the recurrent network by taking advantage of the universal computing properties of recurrent neural networks. Lastly, a proposal for a way to achieve biologically plausible and online reinforcement learning for motor control is presented, based on recently proposed approximations to backpropagation through time (BPTT). Overall, this thesis provides a first direction for understanding how networks of spiking neurons in biology compute, while at the same time remaining functionally capable of solving complex tasks.

Zusammenfassung

Das menschliche Gehirn besitzt eine Vielzahl elementarer Fähigkeiten. Dazu zählen verschiedene höhere kognitive Fähigkeiten, das Erlernen von komplexen Bewegungsabläufen, die Fähigkeit die Struktur der Umgebung zu analysieren und sich dementsprechend an diese anzupassen. Es ist ein ungelöstes Problem, wie diese Fähigkeiten mit Hilfe von Modellen reproduziert werden können, insbesondere im Hinblick auf die biologischen Einschränkungen von Plastizität und Lernvorgängen. Bislang unklar ist, welche induktiven Verzerrungen (inductive biases) und Lernregeln zum Bewältigen dieser Probleme nötig sind und welche neuronalen Repräsentationen sich dabei ergeben. Es ist bekannt, dass sich das Gehirn schnell Adaptieren kann. Da aber kaum biologische Plastizitätsmechanismen bekannt sind, die auf solchen schnellen Zeitskalen ablaufen, wurden Anpassungsmechanismen bislang nur im Rahmen von synaptischer Plastizität erforscht. Diesen Problemen widmet sich die vorliegende Arbeit. Zunächst wird eine induktive Verzerrung für das Lösen von flexiblen kognitiven Entscheidungsaufgaben mittels rekurrenten spikenden neuronalen Netzwerken (RSNNs) untersucht: die Anpassung der Feuerrate. Danach wird beschrieben, wie durch Optimierungsvorgänge, im Gegensatz zu händischer Wahl, weitere induktive Verzerrung in RSNNs installiert werden können. Damit wird demonstriert, dass ein schneller Lern-/Anpassungsvorgang in RSNNs allein durch die Plastizität in einem Readoutmechanismus erfolgen kann, oder sogar komplett ohne synaptische Veränderungen. Diese Art von Anpassung ist auf sehr kurzen Zeitskalen möglich, da sie nicht von den langsamen Abläufen der synaptischen Plastizität im biologischen Kontext beschränkt wird: Anpassungen erfolgen nur innerhalb der Dynamik des rekurrenten Netzwerkes durch die universellen Berechnungseigenschaften solcher Netzwerke. Außerdem wird eine Methode für biologisch plausibles online reinforcement learning in Kontext der motorischer Kontrolle vorgestellt, welches auf einer kürzlich veröffentlichten Approximation des Backpropagation-through-time (BPTT) Algorithmus basiert. Damit stellt diese Arbeit einen ersten Ansatz zur Erklärung von funktionalen spikenden neuronalen Netzen in der Biologie dar.

Acknowledgements

I would like to thank my supervisor Wolfgang Maass for giving me the opportunity to work on very interesting scientific problems, and for his support and guidance during my time as a PhD student. I would also like to thank Sepp Hochreiter for taking the time to be the second referee of this thesis, and for inspiring a lot of the work in this thesis. In addition, I would like to thank: Robert Legenstein for his valuable inputs and support during our various joint projects; David Kappel for all the informative research discussions, and patient explanations, and through whom I learnt a vast amount both about research and otherwise. I would also like to thank the administrative staff at our institute: Daniela Windisch-Scharler, Charlotte Rumpf, Oliver Friedl and Nicoletta Kähling for their constant cheerful support in all administrative matters. I would like to thank all my other current and former colleagues at IGI, including Zeno Jonke, Arjun Rao, Darjan Salaj, Ceca Kraisnikovic, Franz Scherr, Elias Hajek, Guillaume Bellec, Michael Müller, and Florian Unger for various fascinating discussions and productive collaborations. I would like to thank Robert Legenstein, Michael Müller, and Florian Unger for helping me translate my thesis abstract to German; Ceca Kraisnikovic and Ashwini Pandeshwar for proofreading my thesis and their attention to detail.

Most important of all, I would like to thank my wife Ashwini for being an infallible and unending source of love, support, wisdom and cheer through the vagaries of life; my brother Aravind and my parents, Subramoney and Bagavathi, without whom I would not be who I am today.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

The human brain is capable of an astounding variety of complex and subtle behaviour — all the way from fast but precise control of various muscles to slow, complex reasoning, processing, and creation of abstract ideas. The basis of its amazing flexibility is provided by millennia of evolutionary optimizations, and years of developmental processes. This flexibility is also expressed by the brain's ability to learn over time spans of anywhere from years to seconds.

To understand how these processes might work, one common strategy is to build computational and functional models that mimic various aspects of biology Levenstein et al., 2020. Recurrent networks of neurons (RNNs) have been frequently studied as models for networks of neurons in the brain. Specifically, recurrent networks with more biologically plausible and detailed models of neurons — recurrent networks of spiking neurons (RSNNs) — have been used to understand and simulate memory and learning in biological brains Maass, 1997. While RNNs and RSNNs are theoretically powerful models Siegelmann and Sontag, 1995; Maass and Markram, 2004, training them poses its own set of challenges Pascanu et al., 2013. This indicates the necessity of imbuing these networks with inductive biases Mitchell, 1980.

Recurrent networks are often trained using backpropagation through time (BPTT) Werbos, 1990. BPTT has been adapted to work with spiking neurons Bellec et al., 2018b by using a pseudo-derivative for the spiking function that is otherwise not differentiable. These RSNNs can be imbued with memory using spike frequency adaptation (SFA) using adaptive thresholds, making their performance comparable to that of the state-of-the-art models of non-spiking artificial recurrent neural networks — Long short-term Memory (LSTM) networks. Such RSNNs with SFA are called Long short-term memory Spiking Neural Networks (LSNNs). Inductive

bias for fast learning can also be installed in LSNNs using the spiking variation of BPTT using learning-to-learn (L2L) or meta-learning. Since it is not clear how the brain could implement BPTT, biologically plausible approximations to BPTT have been developed Bellec et al., 2019b.

This thesis bases itself on these advances and focuses on exploring three important themes — abstract cognitive tasks, learning to learn, and biologically plausible reinforcement learning. More specifically, these themes translate into achieving and analysing cognitive processing, exploring fast learning without synaptic plasticity through learning to learn, and learning from reward with local synaptic plasticity. They also aim to provide insights into what is possible to achieve with spiking neural networks, and more importantly, how these can be connected with biological models of the brain.

1.1 Cognitive tasks

The ability to perform abstract cognitive computations underlies most definitions of intelligence in the context of artificial intelligence. Many such cognitive tasks can be formulated as operations on abstract sequences of symbols. Humans are not only able to perform such operations, but are also able to do this on strings they have never seen before. This ability to apply rules and perform operations on previously unseen strings of symbols — free generalization — is an essential property of cognitive architectures Marcus, 2003. Some of these tasks are also used for evaluating the abstract reasoning capability of human subjects MacDonald III, 2008. Using neural network models to solve such tasks Rougier et al., 2005; O'Reilly and Frank, 2006 can provide insights into what aspects of the architecture are important, and what mechanisms the network uses to arrive at solutions. As we show here, using neural network models that are based on simple and generic but biologically realistic architectures for these tasks provides interpretable insights into the inductive biases that play an important role in these tasks. The specific biases we study here are for short-term memory, and network codes that emerge for solving such cognitive tasks.

1.2 Learning to learn

Learning to learn or meta-learning is a paradigm where optimization is used to generate a model that can learn tasks from a given family of tasks, rather than constructing and training separate model for each task Thrun and Pratt, 1998; Schmidhuber, 1987. One way this can be achieved is by optimizing a set of hyper-parameters of the model so that the model can learn to solve any of these tasks. In many cases, we want to be able to leverage the common structure present in the family of tasks to learn each task faster compared to a tabula rasa model. Therefore,

a natural setup is to optimize the model hyper-parameters not only learn any task in the family, but also to learn each task quickly Hochreiter et al., 2001.

Most biological brains have strong inductive biases installed over millennia of evolution and years of developmental processes Zador, 2019. Most animals already have the ability to execute a wide repertoire of behaviour required for their survival from the moment they are born. For example, a giraffe can walk within hours of birth, and most insects are born ready to perform all adult behaviours. Additionally, many animals are capable of adapting to the changing world very quickly by taking advantage of and using their prior knowledge of the world to deal with novel stimuli and generate novel behaviours Harlow, 1949. But it is not clear what sort of learning mechanism or plasticity, if at all, is used for implementing this sort of fast learning in biology.

With the observation that recurrent neural networks are theoretically capable of implementing any algorithm in their dynamics Cotter and Conwell, 1990, we study if this could provide the basis for a biologically plausible mechanism for fast learning in biology. Such a model can be implemented in any recurrent network with some form of short-term memory — the shorter memory demands being satisfied with fading memory Maass et al., 2004; Maass et al., 2002, and the longer ones with explicit working memory mechanisms using SFA with adaptive thresholds Bellec et al., 2018b; Salaj et al., 2020. Therefore, such a model would be biologically plausible: it would work with any recurrent network irrespective of neuron or synapse models used, as long as some memory mechanism was installed. The hyper-parameters of the model being optimized for fast learning would be the weights of the network itself, since the fast learning occurs in the dynamics of the network.

In this thesis, we show that such a biologically plausible model for fast learning can be installed using L2L. This model can explain a wide variety of behaviours from motor control to navigation, and is capable of performing complex non-linear tasks. In addition, this can be used to install inductive biases through optimization in the form of priors based on the common structure of the task in the given task family. The network can then use these priors to provide much more robust learning that is not affected by deceptive local task structure, since the network has already learnt the global task structure.

1.3 Reinforcement Learning

The standard way reinforcement learning (RL) is usually formulated is that an agent interacts with an environment by observing the state of the environment, and performing actions that can change its state. Certain transitions lead to a reward, and the agent's goal is to maximize the amount of reward it receives. The agent is not directly provided information about which actions and states are desirable, but rather it must try out different strategies to discover which ones can lead to

rewards. In a general setting, actions may not immediately affect rewards — they can either be delayed, or the environment could require a long sequence of precise actions to lead to a state which provides a reward. A very detailed introduction to the field is provided in R. S. Sutton and Barto, 2018.

Multi-layer "deep" feedforward and recurrent neural networks are commonly used to learn models for such agents, and this combination is referred to as deep reinforcement learning. Deep RL has been responsible for some of the biggest breakthroughs in AI in recent times, starting from DQN Mnih et al., 2015a in 2015 that could play ATARI 2600 games at human level to MuZero Schrittwieser et al., 2020 in 2020 that can play multiple, very different, games from scratch without the rules of the game even being encoded in the learning. Among these, one class of simple, but powerful algorithms that emerged was a range of so-called advantage actor-critic methods Mnih et al., 2016.

A lot of conceptual inspiration in modern deep learning comes directly from biology or is closely connected to biology, ranging from temporal difference error to episodic replay. While deep learning is not directly concerned with developing models that are biologically plausible, such models have the potential to provide validation of biologically plausible learning algorithms and insights into reinforcement learning in biology. Additionally, algorithms that share properties with biological models, such as learning restricted to locally available quantities, and sparse communication and activation can also serve as efficient implementations of these RL algorithms in neuromorphic hardware. The last chapter in this thesis addresses this problem of biologically plausible reinforcement learning in recurrent spiking neural networks.

1.4 Spiking neuron models

In this thesis, we exclusively explore models consisting of networks of spiking neurons Maass, 1997. These neurons are modelled after neurons in the mammalian brain, which communicate with each other using release of neurotransmitters across connections called synapses. This release of neurotransmitters is usually triggered by sharp pulse-like changes in the membrane potential of a neuron called action potentials or "spikes". Such incoming signals lead to membrane-potential depolarization in the receiving neuron, which can trigger another spike if the depolarization crosses a certain threshold. These spikes are binary events — all or none — and are independent of the number and magnitude of incoming signals Barnett and Larkman, 2007.

Early spiking neuron models, such as the Hodgkin-Huxley model Hodgkin et al., 1952, were modelled after detailed biophysical mechanisms in the neuron. Later simplifications of these models based on their abstract and essential properties led to models such as the leaky integrate and fire model Gerstner and Kistler, 2002. A leaky integrate and fire (LIF) model models all incoming signals as currents, and

the membrane potential of the neuron is a linear sum of these currents weighted by the synaptic weights, with an additional leak current. When the membrane potential crosses a certain threshold, the neuron emits a binary spike that is conveyed to the other neurons connected to it. In this thesis, we always use LIF neuron models, and sometimes (when mentioned) extend it with the SFA mechanism described in Chapter 2.

1.5 Organization of the thesis

All results in this thesis are based on publications to which I have contributed as first- or co-first author during my PhD studies. A detailed statement about author contributions are given at the beginning of every chapter. Each chapter consists of the main results and discussion, while the methods and supplemental details are given in the appendices at the end of this thesis.

In Chapter 2, we first address the problem of how spiking neural networks can be trained using BPTT, propose an architecture to imbue spiking neurons with long short-term memory, and train and analyse representations that emerge in such spiking networks in cognitive tasks. The short-term memory in these networks is implemented using spike-frequency adaptation (SFA), a commonly observed property of biological neurons, which was first proposed in Bellec et al., 2018b. We are able to solve cognitive tasks that require two different levels of working memory, and tasks that involve operations on strings of symbols. We also demonstrate that commonly observed phenomena of neural activity — assembly sequences and mixed selectivity — emerge through training in these networks. In Chapter 3, we use a reservoir model to study how learning to learn can be used in networks of spiking neurons. We optimize the weights of a recurrent network of spiking neurons, used as a liquid state machine or a "liquid", to improve its performance on a family of complex non-linear signal processing tasks — predicting the result of applying randomly chosen second-order volterra filters to inputs. Using learning to learn, we generate liquids that can perform significantly better than a random liquid. We show that we can even fix the weights of the readouts, which are otherwise usually trained in a liquid paradigm, and achieve fast learning for a complex non-linear regression task. Overall, this allows us to generate liquids that are capable of much better performance and faster learning than random liquids. In principle, these liquids can then be implemented on an underlying hardware substrate optimized for spiking neurons. In Chapter 4, we study how we can use RSNNs with short-term memory introduced in Chapter 2 to perform biologically realistic tasks and connect it with known phenomena in biology. We demonstrate that such networks can perform fast motor prediction, and solve a navigation task in ways very similar to that seen in biology. Moreover, we demonstrate how prior knowledge about the structure of tasks can be installed in such networks, and the advantages arising out of installing such priors. In Chapter 5, we shift our focus to online reinforcement learning — where learning occurs within biological constraints of

temporal causality and spatial locality. We show how online reinforcement learning can be used to solve a motor control problem using a biologically realistic plasticity rule.

1.6 Publications not included in this thesis

In Rao et al., 2020, we developed a theoretical model for plasticity of layer 5 pyramidal cells based on its properties known from neuroscience — specifically the fact that temporally extended calcium mediated plateau potential leads to burst firing and opening of plasticity windows. Such a neuron model can be shown to perform logistic regression using a very simple plasticity rule, whose dynamics is supported by biological data. The manuscript for this paper is in preparation. I contributed to the conceptualization of the model and the theoretical derivation. In Subramoney and Maass, 2020, we investigated the role of critical dynamics for working memory in networks of spiking neurons. We draw a connection between two measures of criticality used in different fields — lyapunov exponent and avalanches. I contributed to the development of the model, performed the simulations and analysed the simulation data in this model as well. The manuscript for this paper is in preparation.

Chapter 2

Spike-frequency adaptation provides a long short-term memory to networks of spiking neurons

Contents

Abstract. Brains are able to integrate memory from the recent past into their current computations, seemingly without effort. This ability is critical for cognitive tasks such as speech understanding or working with sequences of symbols according to dynamically changing rules. But it has remained unknown how networks of spiking neurons in the brain can achieve that. We show that the presence of neurons with spike frequency adaptation makes a significant difference: Their inclusion in a network moves its performance for such computing tasks from a very low level close to the level of human performance. While artificial neural networks with special long short-term memory (LSTM) units had already reached such high performance levels, they lack biological plausibility. We find that neurons with spike-frequency adaptation, which occur especially frequently in higher cortical areas of the human brain, provide to brains a functional equivalent to LSTM units.

Acknowledgments and author contributions. This chapter is based on the manuscripts

> Darjan Salaj*, Anand Subramoney*, Ceca Kraisnikovic*, Guillaume Bellec, Robert Legenstein, Wolfgang Maass (2020). "Spike-frequency adaptation provides a long short-term memory to networks of spiking neurons." *Submitted for publication. biorXiv:10.1101/2020.05.11.081513*.

To this study, I contributed as first author together with DS and CK. The study was conceived by DS, AS, GB, WM. The experiments were designed by DS, AS, GB, RL, WM and were conducted by DS, AS, CK. The manuscript was written by DS, AS, CK, RL, GB, WM.

2.1 Introduction

Our brains are able to constantly process new information in the light of recent experiences and dynamically changing rules, seemingly without any effort. But we do not know how networks of spiking neurons (SNNs) in the brain accomplish that. The performance of both spike-based and rate-based models for recurrent neural networks in the brain have stayed on a rather low performance level for such tasks, far below the performance level of the human brain and artificial neural network models. Artificial neural network models that perform well on such tasks use, instead of neuron models, a special type of unit called Long Short-Term Memory (LSTM) unit. LSTM units store information in registers — like a digital computer — where it remains without perturbance by network activity for an indefinite amount of time, until is is actively updated or recalled. Hence these LSTM units are not biologically plausible, and it has remained an open problem how neural networks in the brain achieve so high performance on cognitively demanding tasks that require integration of information from the recent past into current computational processing. We propose that the brain achieves this — at least for some tasks — without separating computation and short-term memory in different network modules: Rather it intertwines computing and memory with the help of inherent slow dynamic processes in neurons and synapses.

Arguably the most prominent internal dynamics of neurons on the time scale of seconds — which is particularly relevant for many cognitive tasks — is spike-frequency adaptation (SFA). It is expressed by a substantial fraction of neocortical neurons Allen Institute, 2018. SFA reduces the excitability of a neuron in response to its firing, see Fig. 2.1. Neurons with SFA have often been included in SNN models that aim at modelling the dynamics of brain networks Gutkin and Zeldenrust, 2014, but not in computational studies. We show that neurons with SFA do in fact significantly enhance the computational power of SNNs. This is somewhat surprising, because on first sight their history dependence, which even varies strongly from neuron to neuron, tends to obstruct — rather than enhance — network computations. We propose that this may hold for hand-constructed circuits, whereas evolutionary and learning processes are able to exploit advantages

of such diverse forms of SFA. We demonstrate this in SNN models for a series of demanding benchmark tasks for network computations that all require integration of information over time: Recalling features of fleeting sensory inputs, speech recognition, time series classification, and operations on sequences of symbols.

We also compare the performance of SNNs with SFA to the performance of SNNs that have a different type of slow hidden dynamics, although on a smaller time scale — short-term plasticity (STP) of synapses. But the contribution of synaptic short-term plasticity — especially synaptic facilitation — to computational performance turns out to be lower. Interestingly, the most common form of STP in synapses between pyramidal cells, synaptic depression, tends to provide better support for such computations than synaptic facilitation. References to the related literature can be found in the Discussion.

Altogether our results suggest that neurons with SFA provide to SNNs a similar performance boost for computations that require a long short-term memory as LSTM units do for artificial neural networks. Hence we refer to SNNs that contain neurons with SFA as Long short-term memory SNNs (LSNNs). Since the term short-term memory is more common in the literature on LSTM networks, but the term working memory is more common in the neuroscience literature, and both appear to refer to the same phenomena, we treat these two notions as synonyms and let their use depend on the context.

SNNs are currently of high interest not only for modelling neural networks of the brain, but also as a computing paradigm for drastically more energy-efficient computer hardware. Hence it is of interest to see that the performance of LSTM networks, and thereby many recent achievements in Artificial Intelligence, can be ported to spike-based computing hardware.

2.2 Experimental data and a simple model for SFA

The SFA of a neuron is usually measured in terms of the gradual increase of interspike intervals in its spike response to a constant input drive. An example for such measurement is the Adaptation Index (AI) that is employed by the Allen Institute Allen Institute, 2018, see Fig. 2.1A for samples of neurons with different AI, Fig. 2.1B for the distribution of AI values, and the Methods for the definition of the AI. These data suggest that the human neocortex has a larger fraction of neurons with SFA than the mouse neocortex, and that within the human brain the frontal lobe has a larger fraction than the temporal gyrus. The analysis of experimental data in Pozzorini et al., 2013; Pozzorini et al., 2015 lead to the conclusion that SFA takes place on multiple time scales, with a history dependence that lasts up to 20 s in neocortical neurons. Various models for adapting neurons have been proposed in Gerstner et al., 2014; Teeter et al., 2018. We employ a very simple model for SFA, the generalized leaky integrate-and-fire model $GLIF_2$ from Teeter et al., 2018; Allen Institute, 2017, which we will refer to as the

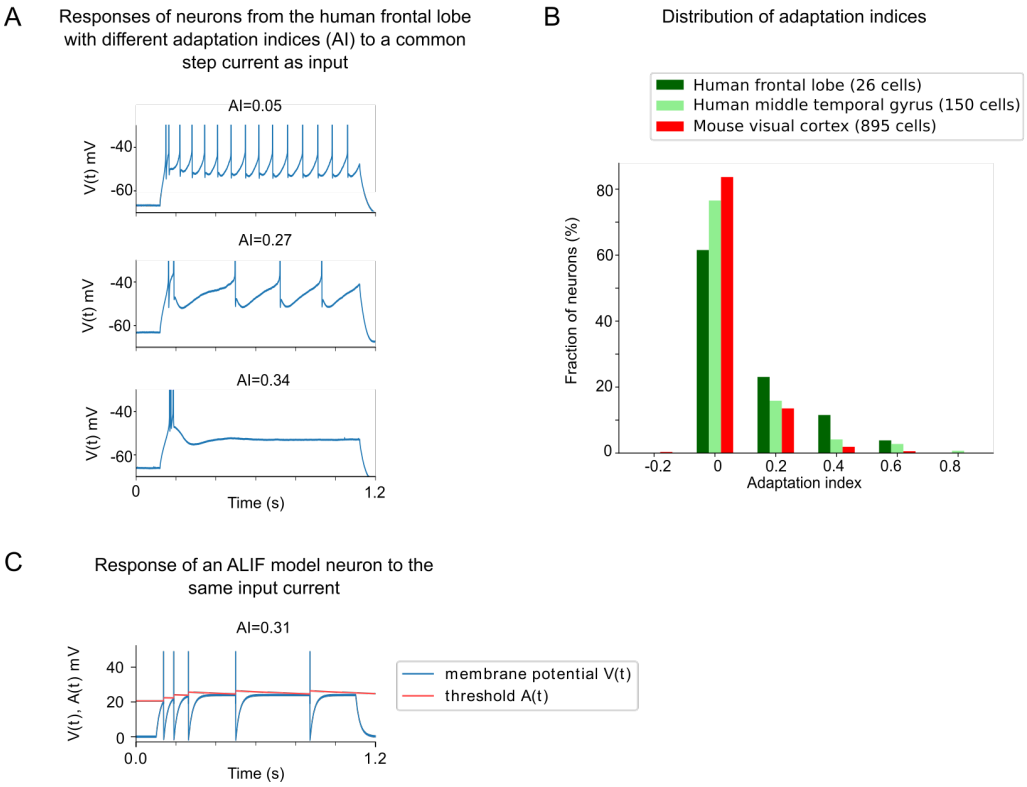

Figure 2.1: Experimental data on neurons with SFA, and a simple model for SFA. (**A**) The response to a 1-second long step current is displayed for three sample neurons in the neocortex. The adaptation index AI measures the rate of increase of interspike intervals. AI > 0 means that a neuron exhibits SFA. (**B**) Distribution of adaptation indices in neurons from human and rodent neocortex. Source of data for A and B: ALLEN INSTITUTE, 2018. (**C**) Response of a simple model for a neuron with SFA — the adaptive LIF (ALIF) model — to the same input current as in A.

ALIF (adaptive LIF) model. A practical advantage of this simple model is that it can be very efficiently simulated and is amenable to gradient descent training methods. It assumes that the firing threshold $A(t)$ contains a variable component $a(t)$ that increases by a fixed amount after each of its spikes $z(t)$ (Fig. 2.1C), and then decays exponentially back to 0. This variable threshold models the inactivation of voltage-dependent sodium channels in a qualitative manner. We write $z_j(t)$ for the spike output of neuron j, that switches from 0 to 1 at time t when the neuron fires at time t, and otherwise has value 0. With this notation one can define the ALIF model by the equations:

$$\begin{aligned} A_j(t) &= v_{\text{th}} + \beta a_j(t), \\ a_j(t+\delta t) &= \rho_j a_j(t) + (1-\rho_j) z_j(t) \delta t, \end{aligned} \tag{2.1}$$

where v_{th} is the constant baseline of the firing threshold $A_j(t)$, and $\beta > 0$ scales the amplitude of the activity-dependent component. The parameter $\rho_j = \exp\left(\frac{-\delta t}{\tau_{a,j}}\right)$ controls the speed by which $a_j(t)$ decays back to 0, where $\tau_{a,j}$ is the adaptation time constant and δt is the duration of a discrete time step (which we chose to be 1 ms). An LSNN is a network of spiking neurons that contains some ALIF neurons (see Methods for details on neuron and synapse models). It typically suffices to use ALIF neurons with some spread of time constants $\tau_{a,j}$ around the required duration of working memory for solving a task (see Table S1 in Supplement for details on how the choice of adaptation time constant impacts performance).

2.3 Methods for training recurrent SNNs

We focused on the best performing training method for recurrent SNNs that is currently known: Backpropagation through time (*BPTT*) with the pseudo-derivative for spiking neurons from Bellec et al., 2018b. While *BPTT* is not assumed to be biologically plausible as an online learning method, results from training with *BPTT* inform us about computational capabilities of different types of SNNs. They also inform us about performance levels that could in principle be attained through evolution. In order to test whether complex cognitive computations, such as the 12AX task , can in principle also be learnt by brain networks, we also trained the same LSNN with a biologically plausible learning method: *e-prop*. For LSNNs, *e-prop* tends to achieve in general an almost as good computational performance level as *BPTT* Bellec et al., 2019b.

2.4 SFA provides a functionally powerful working memory for spike-based computing

Our brains are able to recall an image, even after having seen many other images in between. We wondered whether LSNNs would be able to model such funda-

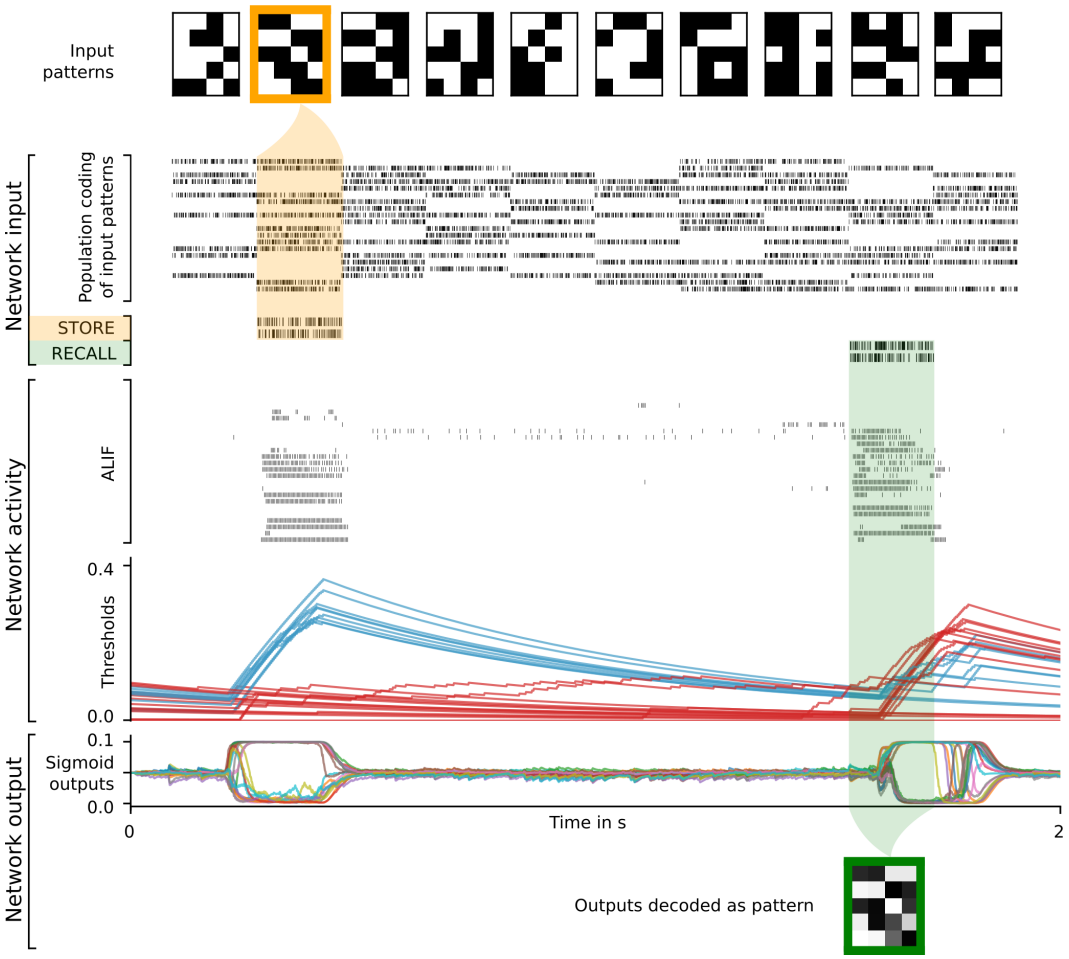

Figure 2.2: High-dimensional working memory capability of an LSNN. Rows top to bottom: Stream of randomly drawn 20 dimensional input patterns, represented by the firing activity of 20 populations of input neurons (subsampled), firing activity of two additional populations of input neurons for the STORE and RECALL commands, firing activity of 25 sample ALIF neurons in the LSNN (we first ordered all ALIF neurons with regard to the variance of their dynamic firing thresholds, and then picked every 20th), temporal evolution of the firing thresholds of these 25 neurons, traces of the activation of 20 sigmoidal readout neurons, and their average value during the 200 ms time window of the RECALL command represented by grey values. During the RECALL command (green shading) the network successfully reproduced the pattern that had been given as input during the preceding STORE command (yellow shading). Coloring of the threshold traces in blue or red was done after visual inspection to highlight the emergent two disjoint populations of ALIF neurons. The activity of one of them peaks during the STORE command, and provides a negative imprint of the stored pattern during RECALL through a reduced firing response. The other one peaks during RECALL.

mental working memory task. Note that remembering an image requires retaining substantially more than a single bit, even if it is encoded in a highly compressed form in a higher cortical area. In contrast, most models for working memory have focused on retaining just a single bit, and this memory content occurred during training and testing. We formulated our more demanding computational task as the STORE-RECALL task illustrated in Fig. 2.2. The network received a sequence of frames, each consisting of a vector of 20 binary features — arranged in a 4×5 grid (top of Fig. 2.2) — which were presented for 200 ms. Each frame can be seen as corresponding to the compressed representation of an image in a higher visual area such as IT. In addition, the network received occasional STORE and RECALL commands, marked in yellow and green in Fig. 2.2. The STORE command corresponds to directing attention to a particular frame of the input stream. The computational task was to reproduce, during a RECALL command, the feature vector that had been presented during the preceding STORE command. The delay between the STORE and RECALL commands was randomly chosen from the interval between 200 and 1600 ms.

We trained an LSNN that consisted of 500 ALIF neurons, whose firing thresholds had time constants of 800 ms, to solve this task. Sigmoidal readout neurons, one for each of the 20 binary input features, were trained to reproduce the feature value that had been present during STORE. Binary feature values were extracted by rounding the activity of readout neurons at the half-way (100 ms) mark of each 200 ms time window. A sample segment of a test trial is shown in Fig. 2.2, with the activity of input neurons at the top and the activation of readout neurons at the bottom. In order to probe the generalization capability of the LSNN we made sure that none of the patterns shown during testing had occurred during training, and in fact had a Hamming distance of at least 5 bits to all training patterns. Note that previous models for working memory only aimed at storing a single bit, and the model could only be tested for the same content for which it was trained. Here we require that the working memory can be used for content other than what was used during training. The resulting recall performance of the LSNN was 99.09%, i.e., 99.09% of the stored feature vectors were perfectly reproduced during recall. This demonstrates that LSNNs have inherent high-dimensional working memory capabilities. SFA was essential for this, because the recall performance of a recurrent network of LIF neurons without SFA, trained in exactly the same way, stayed at chance level (see Methods). A closer inspection of the time course of firing thresholds of a sample subset of neurons in the LSNN provides insight into how LSNNs are able to solve this task: A pattern-specific subset S of neurons is highly activated during STORE, which raises their firing thresholds (shown as blue curves in Fig. 2.2). Many neurons are activated again during RECALL, but the firing activity of neurons in the subset S remains lower this time, thereby providing a negative imprint of their activation pattern during STORE. Readout neurons can easily be trained to decode these negative imprints, and to reproduce the originally stored pattern.

Interestingly, the firing activity of the network was rather low during the delay be-

tween STORE and RECALL. Furthermore we found that a Support Vector Machine (SVM) was not able to decode the stored feature vector from the firing activity of the LSNN neurons during the delay (the decoding accuracy during the delay was 4.38%, as opposed to 100% decoding accuracy during RECALL; see Methods). Hence the type of working memory that an LSNN exhibits corresponds to the activity-silent form of working memory in the human brain that had been examined in the experiments of Wolff et al., 2017. It had also been shown there that the representation of working memory content changes drastically between memory encoding and subsequent network reactivation during the delay by an "impulse stimulus": A classifier trained on the network activity during encoding was not able to classify the memory content during a network reactivation, and vice versa. The same holds for our LSNN model (see Methods), since the reactivation of the network during RECALL provides a negative, rather than a positive imprint, of the high-dimensional memory content.

We also examined how the time constants of the thresholds of ALIF neurons should be chosen to achieve good performance for a task that requires a particular time span of working memory. We studied this for a variant of the STORE-RECALL task where the time-varying input vector of features was just 1D instead of 20D, but where the expected delays between STORE and RECALL varied between 0.2 to 16 s for different versions of the task. It turned out that good performance did not require a tight coupling between the required length of working memory and the adaptation time constants of ALIF neurons in the LSNN, see Table S1. In particular, good working memory performance was also achieved when the required time span for working memory was substantially larger than these time-constants, suggesting that the network had learned to automatically refresh or stagger the implicit memory in firing thresholds of different neurons. We also verified that good performance for many memory retention time spans could be achieved by a single network with a mixture of time constants of firing thresholds drawn from a uniform or power-law distribution. This suggests that brains can solve working memory tasks for many different retention spans using SFA neurons with a generic spread of time constants.

Finally, we wondered whether the adaptive firing threshold of ALIF neurons affects the autocorrelation function of their firing activity — termed intrinsic timescale in Wasmuht et al., 2018. We tested this for an LSNN consisting of 200 LIF and 200 ALIF neurons that was trained to solve a 1D version of the STORE-RECALL task. It turned out that during the delay between STORE and RECALL these intrinsic time constants were in the same range as those measured in monkey cortex, see Fig. 1C in Wasmuht et al., 2018. Furthermore LIF and ALIF neurons exhibited very similar distributions of these time constants (see Fig. S1), suggesting that these intrinsic time constants are determined largely by their network inputs, and less by the neuron type.

2.5 Working memory performance of variants of SNNs with other slow processes in neurons or synapses

There exist numerous other slow processes on the time scale of seconds in neurons and synapses, that can potentially also enhance network computations on this time scale. We examined the performance of three other candidates besides SFA on a simple version of the STORE-RECALL task:

- LIF neurons whose excitability gets increased through their firing: ELIF neurons
- Depressing short-term plasticity of synapses (STP-D)
- Facilitating short-term plasticity of synapses (STP-F).

The ELIF neuron is a dual version of an ALIF neuron whose excitability is increased through preceding firing, rather than decreased (see Methods). ELIF neurons appear to be particularly suitable for creating a working memory through persistent firing. Facilitating short-term plasticity of synapses also supports that, and was conjectured by Mongillo et al., 2008 to produce a working memory. We also evaluated the performance of depressing short-term plasticity, because this is the standard dynamics of synapses between pyramidal cells Markram et al., 2015. The dynamics of the salient hidden variables in these three models is illustrated in Fig. S2. The performance of corresponding variants of the SNN is shown in Fig. 2.3A for a 1D variant of the STORE-RECALL task from Fig. 2.2, with a delay between STORE and RECALL commands varying between 200 ms and 3600 ms. It turns out that SNNs with ALIF neurons, i.e., LSNNs, learn to solve this task much faster than the other variants of the SNN model, and also reach the highest performance level. Furthermore only the networks with STP-D or ELIF neurons eventually approach reasonably good — although lower — performance levels. We were surprised to see that facilitating short-term plasticity of synapses (STP-F) did not provide the working memory capability needed to solve this task, although we used here a really long time constant for facilitation with a mean of 2000 ms — much larger than the mean of 507 ms that had been found experimentally in Y. Wang et al., 2006 for synaptic connections between pyramidal cells in the PFC. Similar results hold for the time series classification task sMNIST, see Fig. 2.3C and the subsection on sMNIST below.

We also found that replacing ALIF by ELIF neurons reduced the working memory capability of the network for both tasks, see Fig. 2.3A and C. One possible reason is that information that is stored in the firing threshold of a neuron is better protected in the case of an ALIF neuron, since an increased firing threshold suppresses subsequent accidental firing, and hence accidental modifications of the memory that is stored in the firing threshold. In contrast, for an ELIF neuron the information that is stored in the firing threshold is quite vulnerable, since a decreased firing threshold invites accidental firing.

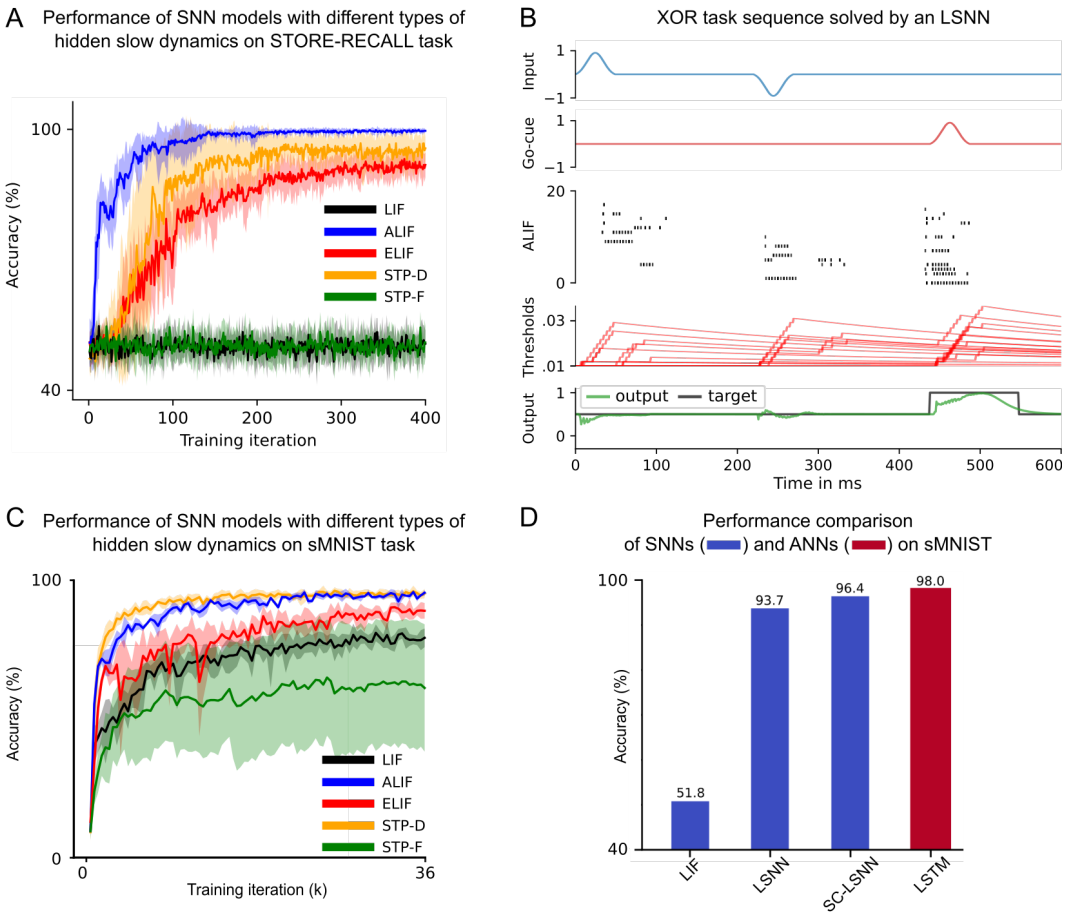

Figure 2.3: Performance comparisons for common benchmark tasks that require substantial integration of information over time. (**A**) Learning curves of networks with different slow processes, for a 1D version of the STORE-RECALL task from Fig. 2.2. The standard SNN ("LIF") as well as SNNs with STP-F cannot learn the task. SNNs with STP-D come closest to the performance level of the LSNN, but require substantially longer training. Mean accuracy and standard deviation are shown for 7 runs with different network initializations for all 5 network types. (**B**) Trained LSNN solving the delayed-memory XOR task HUH and SEJNOWSKI, 2018. Plot of a trial with input consisting of two different types of pulses is shown. From top to bottom: Input pulses, go cue, neuron spike raster, threshold traces, network output. (**C**) Learning curves of five variants of the SNN model (same as in **A**) for the sMNIST time series classification task. Mean accuracy and standard deviation are shown for a minimum of 4 runs with different network initializations for all 5 network types. (**D**) sMNIST performance of two versions of LSNNs are compared with that of an equally large network of LIF neurons, and an LSTM network. SC-LSNN is a sparsely connected LSNN consisting of excitatory and inhibitory neurons.

2.6 Performance of LSNNs for speech recognition and other benchmark tasks that require substantial integration of information over time

Model	test accuracy (%)
FastGRNN-LSQ Kusupati et al., 2018	93.18
LSNN	91.21
LIF network	89.04

Table 2.1: Google Speech Commands. Accuracy of the spiking network models on the test set compared to the state-of-the-art artificial recurrent model reported in Kusupati et al., 2018. Accuracy of the best out of 5 simulations for LSNNs and LIF networks is reported.

Google Speech Commands dataset. We trained LSNNs and networks of LIF neurons on the keyword spotting task with Google Speech Commands Dataset P. Warden, 2018 (v0.02). The dataset consists of 105,000 audio recordings of people saying thirty different words. Fully connected networks were trained to classify audio recordings, that are clipped to one second length, into one of 12 classes (10 keywords, as well as two special classes for silence and unknown words; the remaining 20 words had to be classified as "unknown"). Comparison of maximum performance of trained spiking networks against state-of-the-art artificial recurrent networks is shown in Table 2.1. Averaging over 5 runs, the LSNN reached $90.88 \pm 0.22\%$, and the LIF network reached $88.79 \pm 0.16\%$ accuracy. Thus an SNN without ALIF neurons can already solve this task quite well, but the LSNN halves the performance gap to the published state-of-the-art in machine learning. The only other report on a solution of this task with spiking networks is Cramer et al., 2019. There the authors encode the audio features to spike trains using cochlea model and train a network of LIF neurons using surrogate gradients with *BPTT* and achieve $50.9 \pm 1.1\%$ accuracy.

Delayed-memory XOR task. We also tested the performance of LSNNs on a previously considered benchmark task for SNNs, where two items in the working memory have to be combined non-linearly: The Delayed-memory XOR task Huh and Sejnowski, 2018. The network is required to compute the exclusive-or operation on the history of input pulses when prompted by a go-cue signal, see Fig. 2.3B.

The network receives on one input channel two types of pulses (up or down), and a go-cue on another channel. If the network received two input pulses since the last go-cue signal, it should generate the output "1" during the next go-cue if the input pulses were different or "0" if the input pulses were the same. Otherwise, if the network only received one input pulse since the last go-cue signal, it should generate a null output (no output pulse). Variable time delays are introduced between the input and go-cue pulses. Time scale of the task was 600 ms which limited the delay between input pulses to 200 ms.

This task was solved in Huh and Sejnowski, 2018, without providing a performance statistics, by using a type of neuron that has not been documented in biology — a non-leaky quadratic integrate and fire neuron. We are not aware of previous solutions by networks of LIF neurons. To compare and investigate the impact of SFA on the performance of delayed-memory XOR task, we trained networks of ALIF and LIF neurons of the same size as in Huh and Sejnowski, 2018 — 80 neurons. Across 10 runs, networks of ALIF neurons solved the task with $95.19 \pm 0.014\%$ accuracy, whereas the networks of LIF neurons converged at lower $61.30 \pm 0.029\%$ accuracy.

Sequential MNIST (sMNIST).

Finally, we compared the performance of LSNNs and other variants of SNNs with that of LSTM networks on a more demanding benchmark task for time series classification: The classification of pixel-wise sequentially presented handwritten digits (sMNIST dataset), see Fig. 2.3C,D and Fig. S3. This task requires integration of information over a longer time span than for recognizing speech commands. It also requires very good generalization capability, since the pixel sequences for different handwriting styles of the same digit may vary widely. LSNNs achieved here about the same performance level as LSTM networks, whereas networks that contain only LIF and not ALIF neurons performed poorly, see Fig. 2.3D. Besides a fully connected LSNN, we also tested the performance of a variant of the model, called SC-LSNN, that integrates additional constraints of SNNs in the brain: It is sparsely connected (12% of possible connections are present) and consists of 75% excitatory and 25% inhibitory neurons that adhere to Dale's law. By adapting the sparse connections with the rewiring method in Bellec et al., 2018a during *BPTT* training, the SC-LSNN was enabled to perform even better than the fully-connected LSNN. The resulting architecture of the SC-LSNN is shown in Fig. S3C. Its activity of excitatory and inhibitory neurons, as well as the time courses of adaptive thresholds for (excitatory) ALIF neurons of the SC-LSNN are shown in Fig. S3B.

Fig. 2.3C shows that, apart from LSNNs, SNNs with experimentally reported parameters for short-term synaptic plasticity (STP-D) also achieve very high performance. Furthermore, SNNs with STP-D perform substantially better for this task than networks with data-based synaptic facilitation (STP-F), similar as for STORE-RECALL (see Fig. 2.3A).

2.7 SFA supports demanding cognitive computations with dynamically changing rules

The 12AX task — which can be viewed as a simplified version of the Wisconsin Card Sorting task — tests the capability of subjects to apply dynamically changing rules for pattern recognition and to ignore currently irrelevant inputs O'Reilly

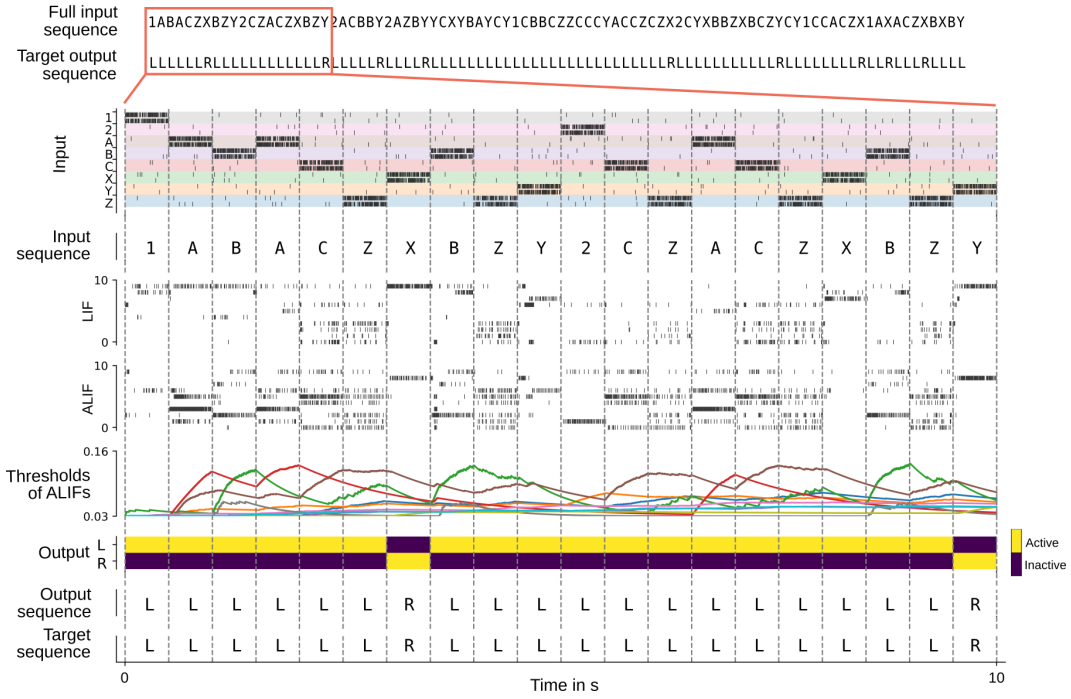

Figure 2.4: Solving the 12AX task by a network of spiking neurons (LSNN), trained with *random e-prop*. A sample trial of the trained network is shown. From top to bottom: Full input and target output sequence for a trial, consisting of 90 symbols each, blow-up for a subsequence of the input symbols, firing activity of 10 sample LIF neurons and 10 sample ALIF neurons from the LSNN, time course of the firing thresholds of these 10 ALIF neurons, activation of the two readout neurons, the resulting sequence of output symbols which the network produced, and the target output sequence.

and Frank, 2006; MacDonald III, 2008. It also probes — at least in the more demanding version that we consider — the capability to maintain and update a hierarchical working memory. The task consists of a sequence of trials where the subject is first shown a context cue to indicate which one of two possible sequences is the "correct" symbol sequence in the current trial. These sequences consist of two relevant symbols, interspersed with distractor symbols, including those belonging to the "wrong" sequence in the current context. At every step, the subject has to press one of two buttons depending on whether the correct sequence has been completed or not. The context of the trial switches randomly after a few presentation of symbols.

To model this, we gave as network inputs sequences of 90 symbols from the set {1, 2, A, B, C, X, Y, Z}, with repetitions as described in Methods. See the top of Fig. 2.4 for an example. After each symbol, the network should output "R" if this symbol terminates a context dependent target sequence and "L" otherwise. Specifically, given a context where the most recently received digit was a "1", the subject should output "R" only after presentation of a symbol X that terminates a subsequence A...X. This occurs, for example, for the 7th symbol in the trial shown in Fig. 2.4. In case that the most recent input digit was a "2", the subject should respond "R" only after the symbol Y in a subsequent subsequence B...Y (see the 20th symbol in Fig. 2.4). The letters C and Z are irrelevant and serve as distractors. This task requires a hierarchical working memory, because the most recently occurring digit determines whether subsequent occurrences of "A" or "B" should be placed into working memory. Note also that neither the content of the higher-level working memory — the digit — nor the content of the lower level working memory — the letter A or B — are simply recalled. Instead, they both affect processing rules, where the higher-level processing rule affects what is placed into the lower level working memory. A simpler version of this task, where X and Y were relevant only if they directly followed A or B respectively, and where fewer irrelevant letters occurred in the input, was solved in O'Reilly and Frank, 2006 through artificial neural network models that were endowed with special working memory modules. Note that for this simpler version no lower order working memory is needed, because one just has to wait for an immediate transition from A to X in the input sequence, or for an immediate transition from B to Y. But neither the simpler nor the more complex version of the 12AX-task has previously been solved by a network of spiking neurons.

We show in Fig. 2.4 that a generic LSNN can solve this quite demanding version of the 12AX task. The LSNN received spike trains from the input population of spiking neurons, producing Poisson spike trains. Possible input symbols {1, 2, A, B, C, X, Y, Z} were encoded using one-hot coding; each input symbol was signaled through a high firing rate of a separate subset of 5 input neurons for 500 ms. The LSNN consisted of 200 recurrently connected spiking neurons (100 ALIF and 100 LIF neurons), with all-to-all connections between them. The output consisted of two readouts, one for L, one for the R response. During each 500 ms time window the input to these readouts was the average activity of neurons in the LSNN during

that time window. The final output symbol was based on which of the two readouts had the maximum value. After training with *BPTT* the LSNN produced an output string with all correct symbols in 97.79% of episodes, where 90 symbols had to be processed during each episode. But also after training with *e-prop*, a biologically realistic learning method Bellec et al., 2019b, the LSNN produced fully correct output sequences in 92.89% of the episodes. In contrast, a recurrent SNN with the same architecture but no neurons with SFA could achieve only 0.39% fully correct output strings after training with *BPTT* (not shown). Note that it was not necessary to create a special network architecture for the two levels of working memory that our more complex version of the 12AX task requires: A near perfectly performing network emerged from training a generic LSNN. This shows that neurons with SFA enable generic recurrent networks of spiking neurons to solve demanding cognitive tasks involving dynamically changing rules and two levels of working memory.

2.8 SFA supports brain-like operations on sequences

A generic difficulty for neural networks is learning to carry out operations on sequences of symbols in such a way that they generalize to new sequences, a fundamental capability of the human brain Marcus, 2003. Actually, not only humans, but also non-human primates are able to carry out operations on sequences of items, and numerous neural recordings — starting with Barone and Joseph, 1989 up to recent results such as Carpenter et al., 2018; Liu et al., 2019 — provide information about the neural codes for sequences that accompany such operations in the brain. One fundamental question is how serial order of items is encoded in working memory. Behind this is the even more basic question of how transient structural information — the serial position of an item — is combined in the brain with content information about the identity of the item Lashley, 1951. Obviously, this question also lies at the heart of open questions about the interplay between neural codes for syntax and semantics that enable language understanding in the human brain. The experimental data both of Barone and Joseph, 1989 and Liu et al., 2019 suggest that the brain uses a factorial code where position and identity of an item in a sequence are encoded separately by some neurons, thereby facilitating flexible generalization of learned experience to new sequences. But so far we had been lacking spiking neural network models that were able to carry out such tasks, and whose emergent neural codes could then be compared with neural recordings from the brain. We show here that LSNNs can be trained to carry out complex operations on sequences, are able to generalize such capabilities to new sequences, and produce spiking activity and neural codes that offer interesting links to recorded data.

One basic operation on sequences of symbols is remembering and reproducing a given sequence Liu et al., 2019, a task which non-human primates can also learn, and for which neural codes for sequences have been investigated Barone and Joseph, 1989; Liu et al., 2019. A more complex operation that can also be carried

out by the brain is the reversal of a sequence Marcus, 2003; Liu et al., 2019. We show that an LSNN learns to carry out both of these operations, and is able to apply them to new sequences of symbols that did not occur during training.

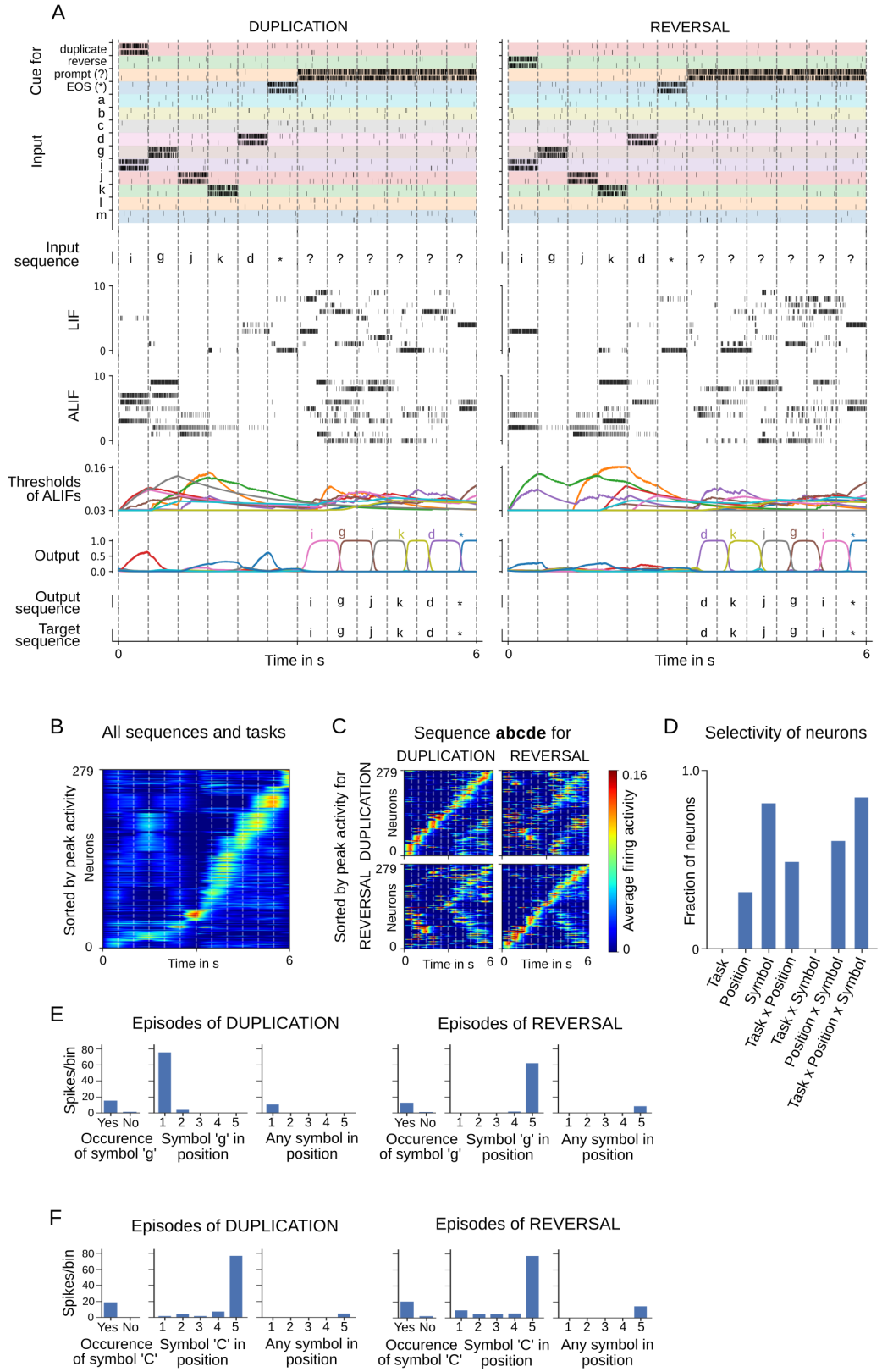

Figure 2.5: (Caption on the next page.)

Figure 2.5: Analysis of an LSNN trained to carry out operations on sequences. (**A**) Two sample episodes where the LSNN carried out sequence duplication (left) and reversal (right). Top to bottom: Spike inputs to the network (subset), sequence of symbols they encode, spike activity of 10 sample LIF, and ALIF neurons in the LSNN, firing threshold dynamics for these 10 ALIF neurons, activation of linear readout neurons, output sequence produced by applying argmax to them, target output sequence. (**B**-**F**) Emergent neural coding of 279 neurons in the LSNN. Neurons sorted by time of peak activity. (**B**) A substantial number of neurons are sensitive to the generic timing of the tasks, especially for the second half of trials when the output sequence is produced. (**C**) Neurons separately sorted for duplication episodes (left column) and reversal episodes (right column). Many neurons respond to input symbols according to their serial position, but differentially for different tasks. (**D**) Histogram of neurons categorized according to conditions with statistically significant effect (3-way ANOVA). Firing activity of a sample neuron that fires primarily when: (**E**) the symbol "g" is to be written at the beginning of the output sequence. The activity of this neuron depends on the task context during the input period; (**F**) the symbol "C" occurs in position 5 in the input, irrespective of the task context.

We trained an LSNN consisting of 128 LIF and 192 ALIF neurons to carry out these two operations on sequences of 5 symbols from a repertoire of 31 symbols, which we labeled by the letters a, b, c, ..., x, y, z, A, B, C, D, E from the English alphabet. Four additional symbols were used: "*" denoted the end of the input sequence (EOS), "?" a prompt for an output symbol, and one symbol each for the DUPLICATE and REVERSE commands (see Fig. 2.5A). Each of the altogether 35 input symbols were given to the network in the form of higher firing activity of a dedicated population of 5 input neurons outside of the LSNN ("one hot encoding"). It was not necessary to assign particular values to adaptation time constants of firing thresholds of neurons with SFA; we simply chose them uniformly randomly to be between 1 ms and 6000 ms, mimicking the diversity of SFA effects found in the neocortex Allen Institute, 2018 in a qualitative manner. The network output was produced by linear readouts (one per potential output symbol, each with a low pass filter with a time constant of 250 ms) that received spikes from neurons in the LSNN, see the row "Output" in Fig. 2.5A). The final output symbol was selected using the readout which had the maximum value at the end of each 500 ms time window (a softmax instead of the hard argmax was used during training), mimicking winner-take-all computations in neural circuits of the brain Chettih and Harvey, 2019 in a qualitative manner.

After training, an LSNN was able to apply duplication and reversal also to new sequences, achieving a success rate of 0.9588 (average over 5 random seeds) for unseen sequences. The "success rate" was defined as the fraction of test episodes/trials where the full output sequence was generated correctly. Sample episodes of the trained LSNN are shown in Fig. 2.5A. For comparison, we also trained a LIF network in exactly the same way with the same number of neurons. It achieved a performance of 0.0 (zero).

Emergent coding properties of neurons in the LSNN are analyzed in Fig. 2.5B-F. Neurons are sorted in Fig. 2.5B,C according to the time of their peak activity (averaged over 1000 episodes), like in Harvey et al., 2012. A number of network neurons (about one-third) participate in sequential firing activity independent

of the type of task and the symbols involved (Fig. 2.5B). Instead, these neurons have learned to abstract the overall timing of the tasks. This kind of activity is reminiscent of the neural activity relative to the start of a trial that was recorded in rodents after they had learned to solve tasks that had a similar duration Tsao et al., 2018.

The time of peak activity of other neurons depended on the task and the concrete content, see Fig. 2.5C. Interestingly enough, these neurons change their activation order already during the loading of the input sequence in dependence of the task (duplication or reversal). Using 3-way ANOVA, we were able to categorize each neuron as selective to a specific condition or a non-linear combination of conditions based on the effect size ω^2. Each neuron could belong to more than one category if the effect size was above the threshold of 0.14 (as suggested by Field, 2013). Similar to recordings from the brain Carpenter et al., 2018, a diversity of neural codes emerged that encode one or several of the variables symbol identity, serial position in the sequence, and type of task. In other words, a large fraction of neurons are mixed-selective, i.e. selective to non-linear combinations of all three variables. Peri-condition time histogram (PCTH) plots of two sample neurons are shown in Fig. 2.5E,F: One neuron is selective to symbol "g" but at different positions depending on task context. The other neuron is selective to symbol "C" occurring at position 5 in the input, independent of task context. Thus one sees that a realization of this task by an SNN, which was previously not available, provides rich opportunities for a comparison of emergent spike codes in the model and neuronal recordings from the brain.

2.9 Discussion

An important open problem in computational neuroscience is to understand how brains carry out computations that involve not just current cues, but information from the recent past. In fact, brains are able to store not just single bits for subsequent computational use over a time scale of many seconds, but previously experienced images, movie scenes, and dialogues that require a fairly large storage capacity. This problem is usually formulated as a question about the implementation of working memory — or short-term memory — in the brain. But this formulation is somewhat biased against the possibility that computing and short-term memory are so intertwined in neural networks of the brain that it becomes really difficult to separate mechanisms and network modules that hold short-term memory from those that constitute the computational machinery of the network.

There already exists fairly wide agreement that different forms of working or short-term memory can be distinguished in the brain, see e.g. Olivers et al., 2011; Kamiński and Rutishauser, 2019; Masse et al., 2019. Recent experimental data show clearly that, for a highly trained task, discrete attractors of the network dynamics, implemented by persistent firing, hold an intended movement direction in the anterior lateral motor cortex Inagaki et al., 2019. But the question remains

whether the brain uses the same mechanism — especially without extensive training — for holding quickly changing high-dimensional memory content, such as a movie scene, previously read text, or a sequence of images. Neural codes for storing information from a sequences of two images — after extensive training — had been examined in M. R. Warden and Miller, 2007. A complex interaction was found between the memory traces of two sequentially presented images, thereby speaking against an assumption that each is held by a separate discrete attractor in working memory.

Several publications argue that the brain uses a variety of mechanisms for working memory, each with its specific advantages and disadvantages, whose engagement depends on the specific task Olivers et al., 2011; Trübutschek et al., 2017; Kamiński and Rutishauser, 2019; Barbosa et al., 2019; Hu et al., 2020. In particular, Wolff et al., 2017; Trübutschek et al., 2017; Kamiński and Rutishauser, 2019; Masse et al., 2019; Barbosa et al., 2019 point to an activity-silent form of working memory that is used by the brain for maintaining working memory while it is not in the focus of attention. A model for such activity-silent memory had been proposed already in Mongillo et al., 2008, based on facilitating short-term plasticity of synapses. This mechanism requires a facilitating short-term plasticity of synapses between excitatory neurons (pyramidal cells), which had previously been discovered in the medial prefrontal cortex of ferret Y. Wang et al., 2006. However the model of Mongillo et al., 2008 used a time constant of 1500 ms for the time constant F of facilitation, whereas this parameter for the facilitation-dominant synapse type E1 of Y. Wang et al., 2006 has a reported average value of 507 ms with a standard deviation of 37 ms. An experimentally testable prediction of this form of activity-silent working memory is that an unspecific network reactivation between storage and recall would make the content of working memory decodable from the resulting network activity. However, the experimental data of Wolff et al., 2017 do not support this prediction.

We examined in this paper whether the arguably most prominent dynamic feature of neurons on the time-scale of seconds, SFA, supports computations that require a working memory. Experimental data show that SFA does in fact produce history-dependence of neural firing on a time scale of several seconds up to 20 seconds Pozzorini et al., 2013; Pozzorini et al., 2015. We found that this prominent feature of a fairly large fraction of neurons in the neocortex provides an inherent working memory capability to neural networks. Our results suggest that this working memory capability is functionally quite powerful, and enables networks of spiking neurons to solve a variety of cognitively demanding tasks that were previously beyond the reach of SNN models. In particular, SFA enables flexible operations on sequences of symbols (Fig. 2.4, 2.5). This allows us, for the first time, to study emergent neural codes for symbols and their position in a sequence in a model network of spiking neurons, and to compare them with recordings from neurons in the neocortex for corresponding tasks (Fig. 2.5). When we compared the contribution of SFA with the contribution of the other two most prominent slow processes in neurons or synapses, synaptic facilitation and synaptic depres-

sion, we found that the contribution of SFA is substantially more powerful for a basic working memory task (Fig. 2.3A). A comparison for a demanding time series classification task with a lower demand on the retention time span (Fig. 2.3C) suggests that synaptic depression works for such tasks about equally well, but not synaptic facilitation. The good performance of synaptic depression for tasks that require shorter retention time of working memory is consistent with the modelling results of Masse et al., 2019 and Hu et al., 2020. However, as already pointed out in Masse et al., 2019, synaptic depression tends to work best for tasks that require rather short working memory maintenance. Our results are also consistent with the finding of Masse et al., 2019 that persistent activity is more prominent if the working memory content has to be manipulated, rather than just maintained. Compare the higher firing activity in Fig. S3B for the sMNIST task that requires continuous manipulation of working memory content with the low firing activity in Fig. 2.2, where the working memory content just has to be maintained. One sees this difference also in the sequence manipulation task of Fig. 2.5. There the working memory just has to be maintained during the first half of a trial, yielding an average firing rate of 16.6 Hz over all neurons. But this average firing rate increased to 26.7 Hz during the second halfs of the trials, where the stored information had to be manipulated (averages taken over 50,000 trials during testing).

On first sight one might think that working memory can only be held in neural activity through increased firing. Our results show that it can just as well be attained through decreased firing, which is the way how neurons with SFA provide evidence of preceding strong activation. Whereas this mechanism may be intuitively less plausible, it looks equally viable from the perspective of downstream networks in the brain. Whether preceding firing activity leaves a positive or negative imprint in subsequent firing appears to be of secondary relevance for readout neurons if the downstream integration of evidence involves a weighted sum, since weights can have positive or negative signs. One may argue that there are actually, from the systems-perspective, two benefits in maintaining working memory in the form of a negative imprint, i.e., through decreased excitability of neurons. One is that encoding working memory through non-firing consumes less energy. Another is that this form of working memory is less vulnerable to disturbances through intervening network activity, since a decreased excitability protects a neuron from accidental activation — and hence potential overwriting of its memory content. One first piece of experimental evidence for the negative imprinting hypothesis was provided by the previously mentioned result of Wolff et al., 2017. It was examined there whether a classifier that had been trained to decode from the network activity the stored memory content during encoding would be able to decode the memory content also during a subsequent network reactivation through an unspecific impulse. The answer was negative, which is consistent with the negative imprinting hypothesis. We confirmed for the task of Fig. 2.2 that the same holds true for our model with SFA (see subsection "Decoding memory from the network activity" in Methods). It is actually well-known that negative imprinting is used by the brain for a particular type of long-term memory called recognition memory: Familiarity of an object is encoded through reduced firing of a large

fraction of neurons in the perirhinal cortex and adjacent areas, see Winters et al., 2008 for a review.

A major structural difference between standard models for neural networks in the brain and artificial neural networks (ANNs) that are used in artificial intelligence and deep learning for solving computational tasks that involve memory from the recent past lies in the type of neurons (units) that are used. Well-performing ANNs usually employ LSTM units or similar units that allow to store a bit or analog variable in a memory register — like in a digital computer — where it is protected from perturbation by ongoing network activity. Our results show — rather surprisingly — that such drastic protection of working memory content is not needed: We showed that almost the same performance can be achieved by LSNNs, i.e., SNNs that contain neurons with SFA. This holds in spite of the fact that memory content that is stored in an adaptive firing threshold of a neuron with SFA is not fully protected from the disturbance through network activity. But it is at least somewhat shielded, because a neuron that holds memory in the form of an increased firing threshold has an inherent tendency not to respond to smaller membrane depolarizations.

Our results show that biologically rather realistic models for spiking neural networks in the brain that also contain neurons with SFA reach for demanding cognitive tasks for the first time the performance level of humans, which could previously only be reached with ANNs that employ biologically unrealistic LSTM units. This paves the way for reaching a key-goal of brain modelling — to combine detailed experimentally data from neurophysiology on the level of neurons and synapses with brain-like functionality of the network.

Recurrent networks of spiking neurons are also of interest from the perspective of novel computing technology. Spike-based computing hardware has the potential to provide substantially more energy-efficient implementations of artificial intelligence and deep learning results than standard digital hardware. But its performance has so far been significantly inferior to that of non-spiking neural networks. Our results show that this performance gap is becoming quite small for the case of recurrent neural networks if one integrates neurons with SFA into the spike-based network.

Altogether, we have shown that a well-known feature of a substantial fraction of neurons in the neocortex — SFA — provides an important new facet to our understanding of computations in SNNs: It enables SNNs to integrate working memory from the recent past seamlessly into ongoing network computations.

Chapter 3

Reservoirs learn to learn

Contents

Abstract. The common procedure in reservoir computing is to take a "found" reservoir, such as a recurrent neural network with randomly chosen synaptic weights or a complex physical device, and to adapt the weights of linear readouts from this reservoir for a particular computing task. We address the question of whether the performance of reservoir computing can be significantly enhanced if one instead optimizes some (hyper)parameters of the reservoir, not for a single task but for the range of all possible tasks in which one is potentially interested, before the weights of linear readouts are optimized for a particular computing task. After all, networks of neurons in the brain are also known to be not randomly connected. Rather, their structure and parameters emerge from complex evolutionary and developmental processes, arguably in a way that enhances speed and accuracy of subsequent learning of any concrete task that is likely to be essential for the survival of the organism. We apply the Learning-to-Learn (L2L) paradigm to mimick this two-tier process, where a set of (hyper)parameters of the reservoir are optimized for a whole family of learning tasks. We found that this substantially enhances the performance of reservoir computing for the families of tasks that we considered. Furthermore, L2L enables a new form of reservoir learning that tends to enable even faster learning, where not even the weights of readouts need to be adjusted for learning a concrete task. We present demos and performance results of these new forms of reservoir computing for reservoirs that consist of networks of spiking neurons, and are hence of particular interest from the perspective of neuroscience and implementations in spike-based neuromorphic hardware. We leave it as an open question what performance advantage the new methods that we propose provide for other types of reservoirs.

Acknowledgments and author contributions. This chapter is based on the manuscripts

> Anand Subramoney, Franz Scherr, Wolfgang Maass (2020). "Reservoirs learn to learn." *Submitted for publication. arXiv:1909.07486*.

To this study, I contributed as first author. The study was conceived by WM, AS. The experiments were designed by WM, AS, FS and were conducted by AS, FS. The manuscript was written by WM, AS, FS.

3.1 Introduction

One motivation for the introduction of the liquid computing model Maass et al., 2002 was to understand how complex neural circuits in the brain, or cortical columns, are able to support the diverse computing and learning tasks which the brain has to solve. It was shown that recurrent networks of spiking neurons (RSNNs) with randomly chosen weights, including models for cortical columns with given connection probability between laminae and neural populations, could in fact support a large number of different learning tasks, where only the synaptic weights to readout neurons were adapted for a specific task Maass et al., 2004; Haeusler and Maass, 2006. Independently from that, a similar framework Jaeger, 2001 was developed for artificial neural networks, and both methods were subsumed under the umbrella of reservoir computing Verstraeten et al., 2007. Our methods for training reservoirs that are discussed in this paper have so far only been tested for reservoirs consisting of spiking neurons, as in the liquid computing model.

Considering the learning capabilities of the brain, it is fair to assume that synaptic weights of these neural networks are not just randomly chosen, but shaped through a host of processes — from evolution, over development to preceding learning experiences. These processes are likely to aim at improving the learning and computing capability of the network. Hence we asked whether the performance of reservoirs can also be improved by optimizing the weights of recurrent connections within the recurrent network for a large range of learning tasks. The Learning-to-Learn (L2L) setup offers a suitable framework for examining this question. This framework builds on a long tradition of investigating L2L, also referred to as meta-learning, in cognitive science, neuroscience, and machine learning Abraham and Bear, 1996; J. X. Wang et al., 2018; Hochreiter et al., 2001; J. X. Wang et al., 2016. The formal model from Hochreiter et al., 2001; J. X. Wang et al., 2016 and related recent work in machine learning assumes that learning (or optimization) takes place in two interacting loops (see Fig. 3.1A). The outer loop aims at capturing the impact of adaptation on a larger time scale (such as evolution, development, and prior learning in the case of brains). It optimizes a set of parameters Θ, for a — in general infinitely large — family $\mathcal{F}$ of learning tasks. Any learning or optimization method can be used for that. For learning a particular task C from $\mathcal{F}$ in the inner loop, the neural network can adapt those of its parameters which do not belong

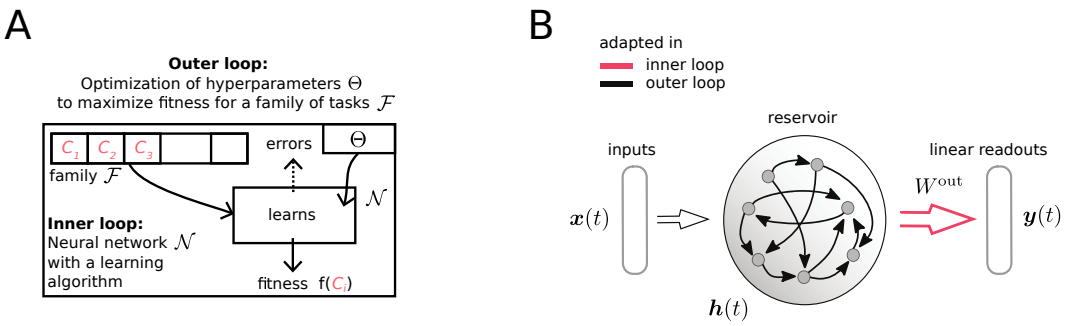

Figure 3.1: Learning-to-Learn setup: **A**) Schematic of the nested optimization that is carried out in Learning-to-Learn (L2L). **B**) Learning architecture that is used to obtain optimized reservoirs

to the hyperparameters Θ that are controlled by the outer loop. These are in our first demo (section 3.2) the weights of readout neurons. In our second demo in section 3.3 we assume that — like in J. X. Wang et al., 2018; J. X. Wang et al., 2016; Hochreiter et al., 2001 — ALL weights from, to, and within the neural network, in particular also the weights of readout neurons, are controlled by the outer loop. In this case just the dynamics of the network can be used to maintain information from preceding examples for the current learning task in order to produce a desirable output for the current network input. One exciting feature of this L2L approach is that all synaptic weights of the network can be used to encode a really efficient network learning algorithm. It was recently shown in Bellec et al., 2018b that this form of L2L can also be applied to RSNNs. We discuss in section 3.3 also the interesting fact that L2L induces priors and internal models into reservoirs.

The structure of this article is as follows: We address in section 3.2 the first form of L2L, where synaptic weights to readout neurons can be trained for each learning task, exactly like in the standard reservoir computing paradigm. We discuss in section 3.3 the more extreme form of L2L where ALL synaptic weights are determined by the outer loop of L2L, so that no synaptic plasticity is needed for learning in the inner loop. Finally, in section 3.4 we will discuss implications of these results, and list a number of related open problems. In the Appendix, we give full technical details for the demos given in sections 3.2 and 3.3.

3.2 Optimizing reservoirs to learn

In the typical workflow of solving a task in reservoir computing, we have to address two main issues: 1) a suitable reservoir has to be generated and 2) a readout function has to be determined that maps the state of the reservoir to a target output. In the following, we address the first issue by a close investigation of how we can improve the process of obtaining suitable reservoirs. For this purpose, we consider here RSNNs as the implementation of the reservoir and its state refers to the activity of all units within the network. In order to generate an instance of such a reservoir, one

usually specifies a particular network architecture of the RSNN and then generates the corresponding synaptic weights at random. Those remain fixed throughout learning of a particular task. Clearly, one can tune this random creation process to better suit the needs of the considered task. For example, one can adapt the probability distribution from which weights are drawn. However, it is likely that a reservoir, generated according to a coarse random procedure, is far from perfect at producing reservoir states that are really useful for the readout.

A more principled way of generating a suitable reservoir is to optimize their dynamics for the range of tasks to be expected, such that a readout can easily extract the information it needs.

Description of optimized reservoirs: The main characteristic of our approach is to view the weight of every synaptic connection of the RSNN that implements the reservoir as hyperparameters Θ, and to optimize them for the range of tasks. In particular, Θ includes both recurrent and input weights (W^{rec}, W^{in}), but also the initialization of the readout $W^{\text{out,init}}$. This viewpoint allows us to tune the dynamics of the reservoir to give rise to particularly useful reservoir states. Learning of a particular task can then be carried out as usual, where commonly a linear readout is learned, for example by the method of least squares or even simpler, by gradient descent.

As previously described, two interacting loops of optimization are introduced, consisting of an inner loop and an outer loop (Fig. 3.1A). The inner loop consists here of tasks C that require to map an input time series $\boldsymbol{x}_C(t)$ to a target time series $\boldsymbol{y}_C(t)$ (see Fig. 3.2A). To solve such tasks, $\boldsymbol{x}_C(t)$ is passed as a stream to the reservoir, which then processes these inputs, and produces reservoir states $\boldsymbol{h}_C(t)$. The emerging features are then used for target prediction by a linear readout:

$$\widehat{\boldsymbol{y}}_C(t) = W_C^{\text{out}}[\boldsymbol{x}_C(t), \boldsymbol{h}_C(t)]^T \,. \tag{3.1}$$

On this level of the inner loop, only the readout weights W_C^{out} are learned. Specifically, we chose here a particularly simple plasticity rule acting upon these weights, given by gradient descent:

$$\Delta W_C^{\text{out}} = \eta \Big(\boldsymbol{y}_C(t) - \widehat{\boldsymbol{y}}_C(t) \Big) \cdot \boldsymbol{h}_C(t)^T \,, \tag{3.2}$$

which can be applied continuously, or changes can be accumulated. Note that the initialization of readout weights is provided as a hyperparameter $W^{\text{out,init}}$ and η represents a learning rate.

On the other hand, the outer loop is concerned with improving the learning process in the inner loop for an entire family of tasks $\mathcal{F}$. This goal is formalized using an optimization objective that acts upon the hyperparameters $\Theta =$

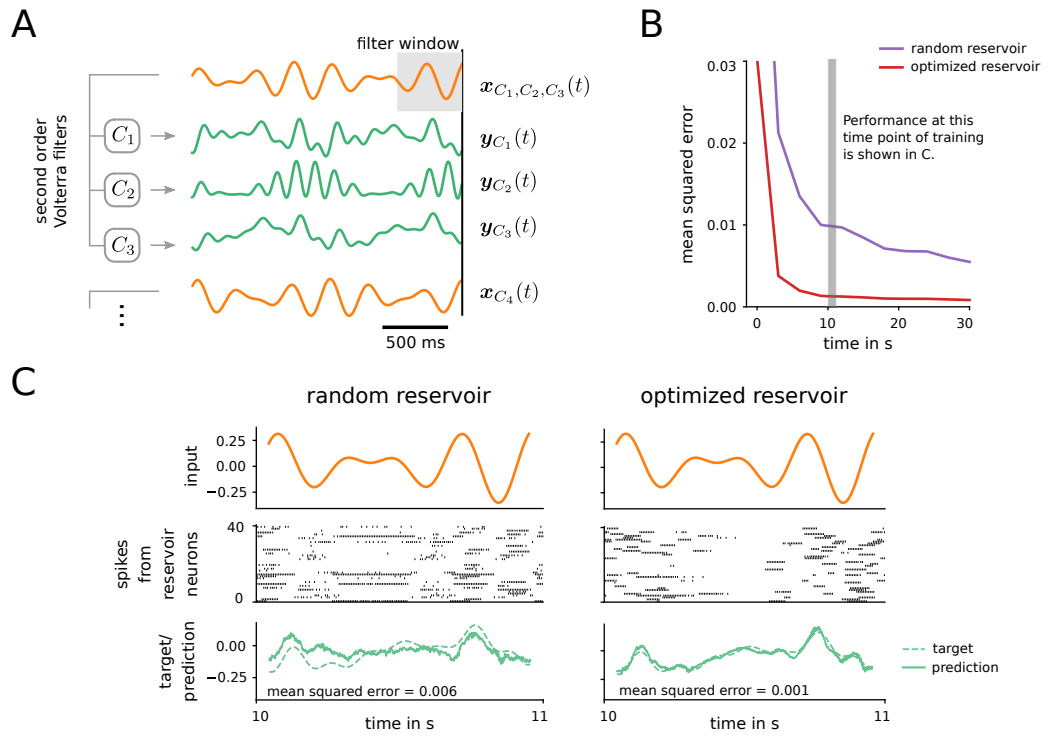

Figure 3.2: Learning to learn a nonlinear transformation of a time series: **A**) Different tasks C_i arise by sampling second order Volterra kernels according to a random procedure. Input time series $x_C(t)$ are given as a sum of sines with random properties. To exhibit the variability in the Volterra kernels, we show three examples where different Volterra kernels are applied to the same input. **B**) Learning performance in the inner loop using the learning rule (3.2), both for the case of a reservoir with random weights, and for a reservoir that was trained in the outer loop by L2L. Performance at the indicated time window is shown in Panel C. **C**) Sample performance of a random reservoir and of an optimized reservoir after readouts have been trained for 10 seconds. Network activity shows 40 neurons out of 800.

$\{W^{\text{in}}, W^{\text{rec}}, W^{\text{out,init}}\}$:

$$\min_{\Theta} \quad \mathbb{E}_{C\sim\mathcal{F}}\left[\int_t \left\|\boldsymbol{y}_C(t) - \widehat{\boldsymbol{y}}_C(t)\right\|_2^2\right] \tag{3.3}$$

$$\text{subject to} \quad \Delta W_C^{\text{out}} = \eta\left(\boldsymbol{y}_C(t) - \widehat{\boldsymbol{y}}_C(t)\right) \cdot \boldsymbol{h}_C(t)^T \quad \text{(readout learning)} \tag{3.4}$$

Regressing Volterra filters: Models of reservoir computing typically get applied to tasks that exhibit nontrivial temporal relationships in the mapping from input signal $x_C(t)$ to target $y_C(t)$. Such tasks are suitable because reservoirs have a property of fading memory: Recent events leave a footprint in the reservoir dynamics which can later be extracted by appropriate readouts. Theory guarantees that a large enough reservoir can retain all relevant information. In practice, one is bound

to a dynamical system of a limited size and hence, it is likely that a reservoir, optimized for the memory requirements and time scales of the specific task family at hand, will perform better than a reservoir which was generated at random.

We consider a task family $\mathcal{F}$ where each task C is determined by a randomly chosen Volterra filter VOLTERRA, 2005. Here, the target $\boldsymbol{y}_C(t)$ arises by application of a randomly chosen second order Volterra filter VOLTERRA, 2005 to the input $\boldsymbol{x}_C(t)$:

$$\boldsymbol{y}_C(t) = \int_\tau k_C^1(\tau)\boldsymbol{x}_C(t-\tau)\,d\tau + \int_{\tau_1}\int_{\tau_2} k_C^2(\tau_1,\tau_2)\boldsymbol{x}_C(t-\tau_1)\boldsymbol{x}_C(t-\tau_2)\,d\tau_1 d\tau_2\ , \quad (3.5)$$

see Fig. 3.2A. The input signal $\boldsymbol{x}_C(t)$ is given as a sum of two sines with different frequencies and with random phase and amplitude. The kernel used in the filter is also sampled randomly according to a predefined procedure for each task C, see Methods C.3, and exhibits a typical temporal time scale. Here, the reservoir is responsible to provide suitable features that typically arise for such second order Volterra filters. In this way readout weights W_C^{out}, which are adapted according to equation (3.2), can easily extract the required information.

Implementation: The simulations were carried out in discrete time, with steps of 1 ms length. We used a network of 800 recurrently connected neurons with leaky integrate-and-fire (LIF) dynamics. Such neurons are equipped with a membrane potential in which they integrate input current. If this potential crosses a certain threshold, they emit a spike and the membrane voltage is reset, see Methods C.1 for details. The reservoir state was implemented as a concatenation of the exponentially filtered spike trains of all neurons (with a time constant of $\tau_{\text{readout}} = 20\,\text{ms}$). Learning of the linear readout weights in the inner loop was implemented using gradient descent as outlined in equation (3.2). We accumulated weight changes in chunks of 1000 ms and applied them at the end. The objective for the outer loop, as given in equation (3.3), was optimized using backpropagation through time (BPTT), which is an algorithm to perform gradient descent in recurrent neural networks. Observe that this is possible because the dynamics of the plasticity in equation (3.2) is itself differentiable, and can therefore be optimized by gradient descent. Because the threshold function that determines the neuron outputs is not differentiable, a heuristic was required to address this problem. Details can be found in the Methods C.2.

Results: The reservoir that emerged from outer-loop training was compared against a reference baseline, whose weights were not optimized for the task family, but had otherwise exactly the same structure and learning rule for the readout. In Fig. 3.2B we report the learning performance on unseen task instances from the family $\mathcal{F}$, averaged over 200 different tasks. We find that the learning performance of the optimized reservoir is substantially improved as compared to the random baseline.

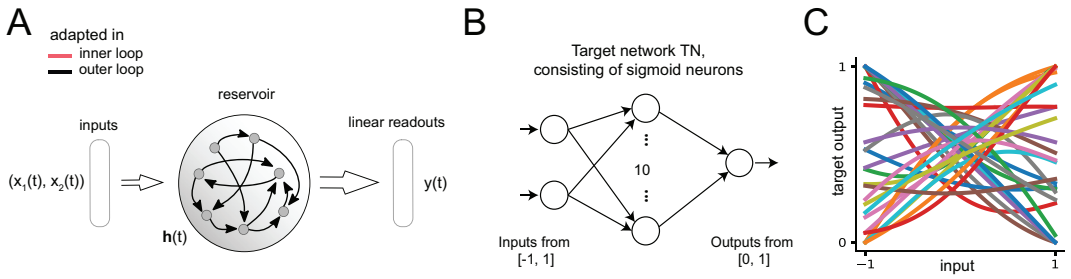

Figure 3.3: L2L setup with reservoirs that learn using their internal dynamics A) Learning architecture for RSNN reservoirs. All the weights are only updated in the outer-loop training using BPTT. **B**) Supervised regression tasks are implemented as neural networks with randomly sampled weights: target networks (TN). **C**) Sample input/output curves of TNs on a 1D subset of the 2D input space, for different weight and bias values.

This becomes even more obvious when one compares the quality of the fit on a concrete example as shown in Fig. 3.2C. Whereas the random reservoir fails to make consistent predictions about the desired output signal based on the reservoir state, the optimized reservoir is able to capture all important aspects of the target signal. This occurred just 10 seconds within learning the specific task, because the optimized reservoir was already confronted before with tasks of a similar structure, and could capture through the outer loop optimization the smoothness of the Volterra kernels and the relevant time dependencies in its recurrent weights.

3.3 Reservoirs can also learn without changing synaptic weights to readout neurons

We next asked whether reservoirs could also learn a specific task without changing any synaptic weight, not even weights to readout neurons. It was shown in Hochreiter et al., 2001 that LSTM networks can learn nonlinear functions from a teacher without modifying their recurrent or readout weights. It has recently been argued in J. X. Wang et al., 2018 that the pre-frontal cortex (PFC) accumulates knowledge during fast reward-based learning in its short-term memory, without using synaptic plasticity, see the text to supplementary Fig. 3 in J. X. Wang et al., 2018. The experimental results of Perich et al., 2018 also suggest a prominent role of network dynamics and short-term memory for fast learning in the motor cortex. Inspired by these results from biology and machine learning, we explored the extent to which recurrent networks of spiking neurons can learn using just their internal dynamics, without synaptic plasticity.

In this section, we show that one can generate reservoirs through L2L that are able to learn with fixed weights, provided that the reservoir receives feedback about the prediction target as input. In addition, relying on the internal dynamics of the

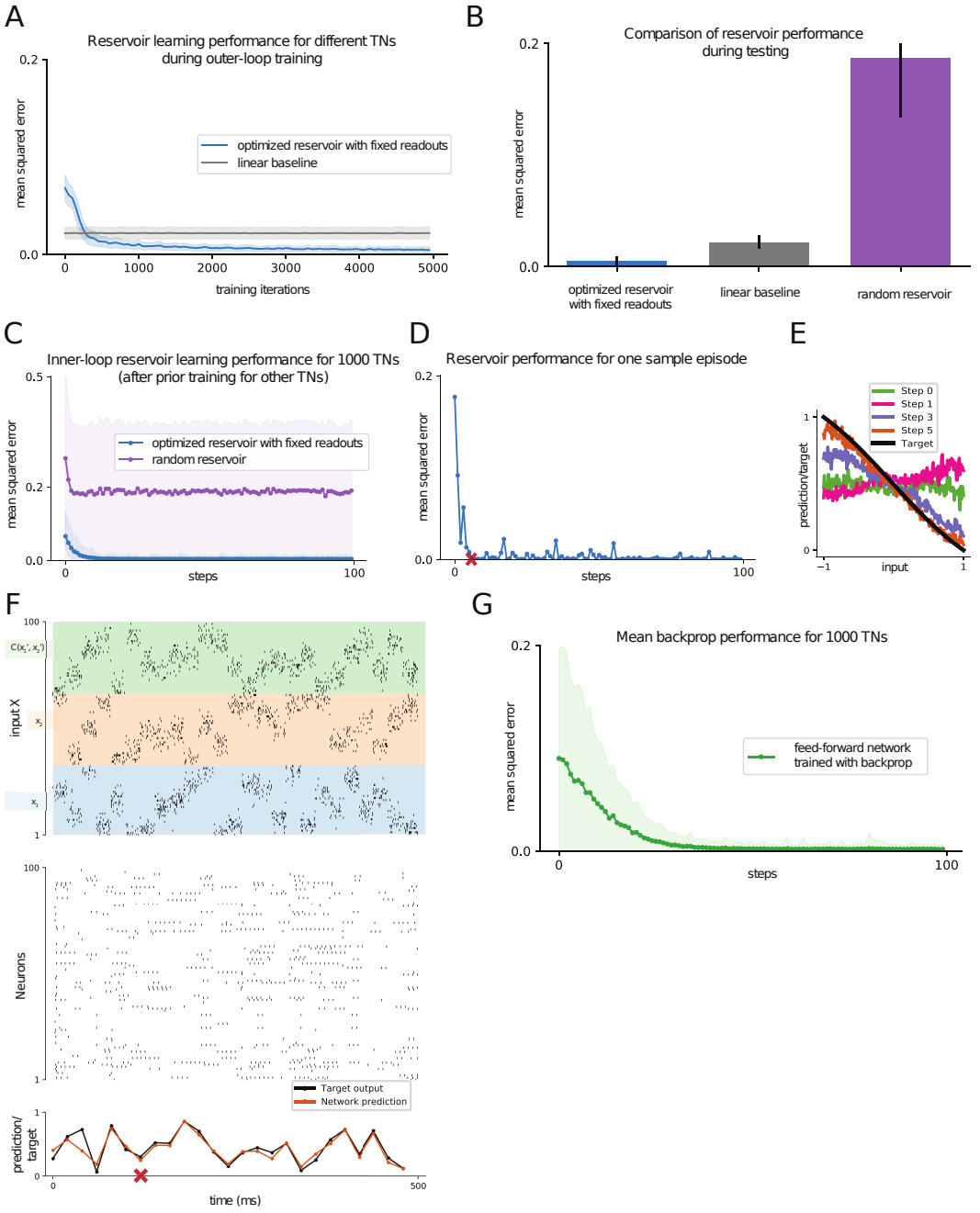

Figure 3.4: (Caption next page.)

Figure 3.4: Learning to learn a nonlinear function that is defined by an unknown target network (TN): A) Performance of the reservoir in learning a new TN during training in the outer loop of L2L. **B**) Performance of the optimized reservoir during testing compared to a random reservoir and the linear baseline. **C**) Learning performance within a single inner-loop episode of the reservoir for 1000 new TNs (mean and one standard deviation). Performance is compared to that of a random reservoir. **D**) Performance for a single sample TN, a red cross marks the step after which output predictions became very good for this TN. The spike raster for this learning process is the one depicted in (F). **E**) The internal model of the reservoir (as described in the text) is shown for the first few steps of inner loop learning. The reservoir starts by predicting a smooth function, and updates its internal model in just 5 steps to correctly predict the target function. **F**) Network input (top row, only 100 of 300 neurons shown), internal spike-based processing with low firing rates in the neuron populations (middle row), and network output (bottom row) for 25 steps of 20 ms each. **G**) Learning performance of backpropagation for the same 1000 TNs as in C, working directly on the ANN from Fig. 3.3B, with a prior for small weights, with the best hyper-parameters from a grid-search.

reservoir to learn allows the reservoir to learn as fast as possible for a given task i.e. the learning speed is not determined by any predetermined learning rate.

Target networks as the task family $\mathcal{F}$: We chose the task family to demonstrate that reservoirs can use their internal dynamics to regress complex non-linear functions, and are not limited to generating or predicting temporal patterns. This task family also allows us to illustrate and analyse the learning process in the inner loop more explicitly. We defined the family of tasks $\mathcal{F}$ using a family of non-linear functions that are each defined by a target feed-forward network (TN) as illustrated in Fig. 3.3B. Specifically, we chose a class of continuous functions of two real-valued variables (x_1, x_2) as the family $\mathcal{F}$ of tasks. This class was defined as the family of all functions that can be computed by a 2-layer artificial neural network of sigmoidal neurons with 10 neurons in the hidden layer, and weights and biases in the range [-1, 1]. Thus overall, each such target network (TN) from $\mathcal{F}$ was defined through 40 parameters in the range [-1, 1]: 30 weights and 10 biases. Random instances of target networks were generated for each episode by randomly sampling the 40 parameters in the above range. Most of the functions that are computed by TNs from the class $\mathcal{F}$ are nonlinear, as illustrated in Fig. 3.3C for the case of inputs (x_1, x_2) with $x_1 = x_2$.

Learning setup: In an inner loop learning episode, the reservoir was shown a sequence of pairs of inputs (x_1, x_2) and delayed targets $C(x_1', x_2')$ sampled from the non-linear function generated by one random instance of the TN. After each such pair was presented, the reservoir was trained to produce a prediction $\hat{C}(x_1, x_2)$ of $C(x_1, x_2)$. The task of the reservoir was to produce predictions with a low error. In other words, the task of the reservoir was to perform non-linear regression on the presented pairs of inputs and targets and produce predictions of low-error on new inputs. The reservoir was optimized in the outer loop to learn this fast and well.

When giving an input x_1, x_2 for which the reservoir had to produce prediction $\hat{C}(x_1, x_2)$, we could not also give the target $C(x_1, x_2)$ for that same input at the same time. This is because, the reservoir could then "cheat" by simply producing this value $C(x_1, x_2)$ as its prediction $\hat{C}(x_1, x_2)$. Therefore, we gave the target value to the reservoir with a delay, after it had generated the prediction $\hat{C}(x_1, x_2)$. Giving the target value as input to the reservoir is necessary, as otherwise, the reservoir has no way of figuring out the specific underlying non-linear function for which it needs to make predictions.

Learning is carried out simultaneously in two loops as before (see Fig. 3.1A). Like in Hochreiter et al., 2001; J. X. Wang et al., 2016; Duan et al., 2016 we let all synaptic weights of $\mathcal{N}$, including the recurrent, input and readout weights, to belong to the set of hyper-parameters that are optimized in the outer loop. Hence the network is forced to encode all results from learning the current task C in its internal state, in particular in its firing activity. Thus the synaptic weights of the neural network $\mathcal{N}$ are free to encode an efficient *algorithm* for learning arbitrary tasks C from $\mathcal{F}$.

Implementation: We considered a reservoir $\mathcal{N}$ consisting of 300 LIF neurons with full connectivity. The neuron model is described in the Methods C.1. All neurons in the reservoir received input from a population X of 300 external input neurons. A linear readout receiving inputs from all neurons in the reservoir was used for the output predictions. The reservoir received a stream of 3 types of external inputs (see top row of Fig. 3.4F): the values of x_1, x_2, and of the output $C(x_1', x_2')$ of the TN for the preceding input pair x_1', x_2' (set to 0 at the first trial), each represented through population coding in an external population of 100 spiking neurons. It produced outputs in the form of weighted spike counts during 20 ms windows from all neurons in the network (see bottom row of Fig. 3.4F). The weights for this linear readout were trained, like all weights inside the reservoir, in the outer loop, and remained fixed during learning of a particular TN.

The training procedure in the outer loop of L2L was as follows: Network training was divided into training episodes. At the start of each training episode, a new TN was randomly chosen and used to generate target values $C(x_1, x_2) \in [0, 1]$ for randomly chosen input pairs (x_1, x_2). 400 of these input pairs and targets were used as training data, and presented one per step to the reservoir during the episode, where each step lasted 20 ms. The reservoir parameters were updated using BPTT to minimize the mean squared error between the reservoir output and the target in the training set, using gradients computed over batches of 10 such episodes, which formed one iteration of the outer loop. In other words, each weight update included gradients calculated on the input/target pairs from 10 different TNs. This training procedure forced the reservoir to adapt its parameters in a way that supported learning of many different TNs, rather than specializing on predicting the output of single TN. After training, the weights of the reservoir remained fixed, and it was required to learn the input/output behaviour of TNs from $\mathcal{F}$ that it had never seen

before in an online manner by just using its fading memory and dynamics. See the Methods C.4 for further details of the implementation.

Results: The reservoir achieves low mean-squared error (MSE) for learning new TNs from the family $\mathcal{F}$, significantly surpassing the performance of an optimal linear approximator (linear regression) that was trained on all 400 pairs of inputs and target outputs, see grey bar in Fig. 3.4B. One sample of a generic learning process is shown in Fig. 3.4D.

Each sequence of examples evokes an "internal model" of the current target function in the internal dynamics of the reservoir. We make the current internal model of the reservoir visible by probing its prediction $C(x_1, x_2)$ for hypothetical new inputs for evenly spaced points (x_1, x_2) in the entire domain, without allowing it to modify its internal state (otherwise, inputs usually advance the network state according to the dynamics of the network). Fig. 3.4E shows the fast evolution of internal models of the reservoir for the TN during the first trials (visualized for a 1D subset of the 2D input space). One sees that the internal model of the reservoir is from the beginning a smooth function, of the same type as the ones defined by the TNs in $\mathcal{F}$. Within a few trials this smooth function approximated the TN quite well. Hence the reservoir had acquired during the training in the outer loop of L2L a prior for the types of functions that are to be learnt, that was encoded in its synaptic weights. This prior was in fact quite efficient, as Figs. 3.4C,D,E show, compared to that of a random reservoir. The reservoir was able to learn a TN with substantially fewer trials than a generic learning algorithm for learning the TN directly in an artificial neural network as shown in Fig. 3.4G: backpropagation with a prior that favored small weights and biases. In this case, the target input was given as feedback to the reservoir throughout the episode, and we compare the training error achieved by the reservoir with that of a FF network trained using backpropagation. A reservoir with a long short-term memory mechanism where we could freeze the memory after low error was achieved allowed us to stop giving the target input after the memory was frozen (results not shown). This long short-term memory mechanism was in the form of neurons with adapting thresholds as described in Bellec et al., 2018b; Salaj et al., 2020. These results suggest that L2L is able to install some form of prior knowledge about the task in the reservoir. We conjectured that the reservoirs fits internal models for smooth functions to the examples it received.

We tested this conjecture in a second, much simpler, L2L scenario. Here the family $\mathcal{F}$ consisted of all sine functions with arbitrary phase and amplitudes between 0.1 and 5. The reservoir also acquired an internal model for sine functions in this setup from training in the outer loop, as shown in Bellec et al., 2018b. Even when we selected examples in an adversarial manner, which happened to be in a straight line, this did not disturb the prior knowledge of the reservoir.

Altogether the network learning that was induced through L2L in the reservoir is of particular interest from the perspective of the design of learning algorithms,

since we are not aware of previously documented methods for installing structural priors for online learning of an RSNN.

3.4 Discussion

We have presented a new form of reservoir computing, where the reservoir is optimized for subsequent fast learning of any particular task from a large – in general even infinitely large – family of possibly tasks. We adapted for that purpose the well-known L2L method from machine learning. We found that for the case of reservoirs consisting of spiking neurons this two-tier process does in fact enhance subsequent reservoir learning performance substantially in terms of precision and speed of learning. We propose that similar advantages can be gained for other types of reservoirs, e.g. recurrent networks of artificial neurons or physical embodiments of reservoirs (see Tanaka et al., 2019 for a recent review) for which some of their parameters can be set to specific values. If one does not have a differentiable computer model for such physically implemented reservoir, one would have to use a gradient-free optimization method for the outer loop, such as simulated annealing or stochastic search, see Bohnstingl et al., 2019 for a first step in that direction.

We have explored in Sec. 3.3 a variant of this method, where not even the weights to readout neurons need to be adapted for learning a specific tasks. Instead, the weights of recurrent connections within the reservoir can be optimized so that the reservoir can learn a task from a given family $\mathcal{F}$ of tasks by maintaining learnt information for the current task in its working memory, i.e., in its network state. This state may include values of hidden variables such as current values of adaptive thresholds, as in the case of LSNNs Bellec et al., 2018b. It turns out that L2L without any synaptic plasticity in the inner loop enables the reservoir to learn faster than the optimal learning method from machine learning for the same task: Backpropagation applied directly to the target network architecture which generated the nonlinear transformation, compare panels C and G of Fig. 3.4. We also have demonstrated in Fig. 3.4E (and in Bellec et al., 2018b) that the L2L method can be viewed as installing a prior in the reservoir. This observation raises the question what types of priors or rules can be installed in reservoirs with this approach. For neurorobotics applications it would be especially important to be able to install safety rules in a neural network controller that can not be overridden by subsequent learning. We believe that L2L methods could provide valuable tools for that.

Another open question is whether biologically more plausible and computationally more efficient approximations to BPTT, such as e-prop Bellec et al., 2019b, can be used instead of BPTT for optimizing a reservoir in the outer loop of L2L. In addition it was shown in Bellec et al., 2019a that if one allows that the reservoir adapts weights of synaptic connections within a recurrent neural network via e-prop, even one-shot learning of new arm movements becomes feasible.

Chapter 4

Prior knowledge and network dynamics enable networks of spiking neurons to learn new tasks from just a few trials

Contents

Abstract. Brains can learn a new task much faster because they can engage prior knowledge to learn a new task with few trials. In fact, it has recently been argued that neural networks in the pre-frontal cortex (PFC) can learn even faster than models based on synaptic plasticity by storing information from prior learning trials in their dynamic state, rather than in synaptic weights. But we are missing methods that could explain how these functionally important fast forms of learning are implemented in neural networks of the brain, rather than in artificial neural network models such as long-short term memory (LSTM) networks. We show here how prior knowledge can be installed in networks of spiking neurons, and how such networks can learn without even engaging synaptic plasticity. Using our biologically realistic network models, we also make experimentally testable predictions and provide neural correlates of fast learning without synaptic plasticity. Fast adaptation is also known to occur in the motor cortex for motor prediction, and we provide a model that proposes a possible mechanism for such rapid adaptation. Moreover, we demonstrate these previously unknown capabilities of spiking neural networks for fast learning for the water-maze task. Since spiking neural networks are also of interest for current work on the design of energy-efficient computing hardware for AI, and since fast learning is essential in many AI applications, our methods also provide new insights for these application domains.

Acknowledgments and author contributions. This chapter is based on the manuscripts

> Anand Subramoney, Guillaume Bellec, Franz Scherr, Robert Legenstein, Wolfgang Maass (2020). "Prior knowledge and network dynamics enable networks of spiking neurons to learn new tasks from just a few trials." *In preparation*.

To this study, I contributed as first author. The study was conceived by WM, AS. The experiments were designed by WM, AS, GB and were conducted by AS, GB, FS. The manuscript was written by AS, WM, GB.

4.1 Biological inspiration for enhancing learning in recurrent networks of spiking neurons

It is known that biological neural networks are capable of multiple levels of learning, including on very fast and slow time scales Botvinick et al., 2019. Fast learning happens much more quickly, and in much fewer trials or learning attempts, than can be explained by synaptic plasticity mechanisms like spike-time dependent plasticity (STDP), and closer to the time scales of short-term memory J. X. Wang et al., 2018. Mechanisms for this fast learning can be installed in the form of inductive biases or priors either through the slower learning process during the lifetime of an organism, or through longer term evolutionary and developmental processes Zador, 2019. These inductive biases or priors facilitate fast learning for specific families of tasks that are important for the organism, and can emerge due to the advantage it provides an organism over using only innate knowledge from evolution or slower learning processes and can be produced by these very processes Zador, 2019. This mechanism of installing this fast learning process, known as learning to learn Harlow, 1949, was demonstrated in experiments with monkeys. Here, a monkey learnt over time that one of two potentially unknown objects they were shown contained a reward and the other did not. These objects were switched every 6 trials to entirely new pairs of objects the monkey had never seen before, but one object was consistently associated with a reward within a block of trials, and the other was not. After some training, the monkeys learnt to infer which object was associated with the reward in just one trial, even when the objects changed in blocks of trials.

While spike time dependent plasticity (STDP) is a well understood and biologically supported learning mechanism, it usually requires $50 - 100$ pairings repeated over several minutes, and evoked at low frequencies of $0.1 - 5$Hz for successful induction of long-term potentiation and depression Froemke et al., 2010. So it is not a plausible candidate for learning in biology that can happen in some situations in the matter of a single or a few trials Harlow, 1949; Behrens et al., 2007; J. X. Wang et al., 2018, where each trial corresponds to one action that receives an associated reward.

Evidence from neuroscience indicates that fast learning can occur using only the dynamics of the brain activity, without synaptic plasticity, in various contexts including reinforcement learning and motor adaptation. It has been suggested J. X. Wang et al., 2018 that the pre-frontal cortex (PFC) implements fast reward-based learning in its activity without using dopamine-gated synaptic plasticity. The dopaminergic learning is rather used for the longer learning process in the PFC that leads to the weights that enable fast learning. In fact, J. X. Wang et al., 2018 show that a recurrent network (LSTM) with fixed weights trained using meta-reinforcement learning qualitatively matches the behaviour from cognitive and biological experiments. In other words, they argue that part of the learning in the pre-frontal cortex can happen even without synaptic plasticity. Rapid adaptation was also demonstrated during motor adaptation Perich et al., 2018 in the absence of any changes in functional synaptic connectivity.

In the domain of machine learning, this fast learning also hints at a solution to the general sample inefficiency of supervised and reinforcement learning methods Zador, 2019; Botvinick et al., 2019. One way of achieving such fast learning, using learning-to-learn, was shown in Hochreiter et al., 2001: Long short-term memory (LSTM) networks were trained using the learning-to-learn framework to do fast learning from supervised examples using only their internal recurrent dynamics. That is, they learned to implement fast learning for specific families of tasks in their recurrent dynamics. This was a practical demonstration of the earlier theoretical result that recurrent networks with fixed weights can learn Cotter and Conwell, 1990. It was further shown in J. X. Wang et al., 2016; Duan et al., 2016 that this result can be extended to reinforcement learning.

All these earlier results use LSTM networks, or LSTMs Hochreiter and Schmidhuber, 1997. LSTMS are a class of networks that are imbued with long short-term memory due to specific architectural choices in the form of various gating units. This makes it feasible to train it to solve tasks that have longer memory requirements. More specifically, each LSTM unit consists of a memory cell that holds real valued information, with gates controlling the flow of input to this cell, output from the cell, and the decay of the contents of the cell itself (forget gate). But these very architectural choices make it biologically very unrealistic in terms of how its computation and memory are implemented, due to the use of multiplicative gates and the separate digital memory cell in every unit.

Since L2L thus far had only been demonstrated with these quite unbiological models of recurrent computation J. X. Wang et al., 2016; J. X. Wang et al., 2018; Hochreiter et al., 2001, it was not clear if the more biologically plausible networks of spiking neurons are capable of learning to implement learning in its dynamics which consist of discrete spiking states. We show that this is possible by harnessing recent innovations in implementing long short-term memory in spiking neural networks using spike frequency adaptation Bellec et al., 2018b; Salaj et al., 2020. This network model, called LSNN (Long-short term memory Spiking Neural Networks) has similar functionality as LSTMs, but is modelled on real biological neurons rather than abstract entities such as memory cells without decay. LSNNs

also well suited for L2L since they can accommodate two levels of learning and representation of learned insight: The synaptic weights can encode, at a higher level, a learning algorithm and prior knowledge on long time scales. The short-term memory of the LSNN can accumulate, at a lower level, knowledge relevant to the current learning task. Some L2L tasks have memory requirements on a very short time scale, where purely recurrent fading memory exhibited by reservoirs Maass et al., 2002; Jaeger, 2001 suffices without additional memory mechanisms: These tasks are demonstrated in Subramoney et al., 2019. But for most biologically realistic tasks like motor prediction and reinforcement learning, memory on longer time scales is required.

Therefore, in this paper, we show that we can implement fast learning in recurrent networks of spiking neurons sans synaptic plasticity. We first demonstrate it on a toy example of learning a sinusoidal function. This example also elucidates a novel property of this framework: It enables installing prior knowledge into networks based on the shared structure of the task, without any manual intervention. These priors give an indication of how the learning can be fast and use only a few examples. Since these priors are learnt through optimization, this has potential applicability in understanding how prior knowledge and inductive biases are learnt in biology. We hypothesize that learning such priors forms a very important factor in enabling fast learning in biology, and we present one experimentally testable prediction in terms of firing rates that are measurable in biology. Additionally, one can apply this framework to a very large variety of machine learning tasks where safety and robustness are paramount. We then provide a concrete demonstration of the utility of this framework in modelling biological phenomena: specifically, how rapid adaptation of motor prediction can occur when changes in limb kinematics occur. This gives a parsimonious explanation to the phenomena observed in Perich et al., 2018 in the limited context of motor prediction (and not control). Finally, we show that networks of spiking neurons can learn to learn on a continuous control reinforcement learning problem formulated in the form of a dynamic water-maze task that is often used in biological experiments. Here, the agent needs to learn to learn how to get to a goal that changes its position every episode. Overall, we propose that being able to learn without using synaptic plasticity could play an important role, and an explanation for how biology achieves fast learning or rapid adaptation in certain scenarios like motor prediction, or adaptive navigation.

4.2 Learning to learn

The idea of learning to learn (L2L) is to optimize the model not for a single task, but instead assume that the model must learn to perform well in a dynamic environment that consists of a series of interrelated tasks. This family of interrelated tasks can, in the general case, be infinitely large. We optimize the model for fast learning of each task chosen from this family of tasks. This means that learning is carried out at two levels (see Fig. 4.1A): The “inner loop” involves the learning

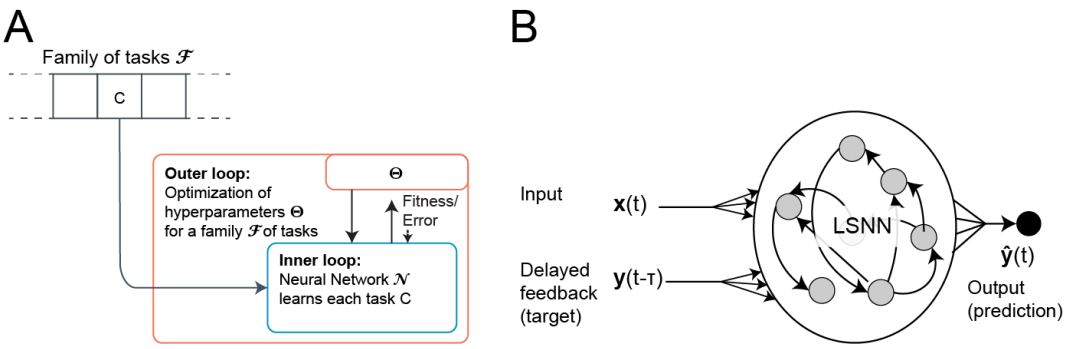

Figure 4.1: Learning to learn (L2L) setup. (**A**) Schematic of the learning to learn setup. (**B**) Common architecture for LSNN with inputs and outputs. All the weights are only updated in the outer-loop training using BPTT, and remain fixed during testing.

of a single task by the network. The "outer loop" involves optimization of some parameters of the network to support fast learning of the individual tasks in the inner loop. The outer loop training proceeds on a much larger time scale than the inner loop, integrating performance evaluations from many different individual tasks. This outer loop can be interpreted as a process that mimics the impact of evolutionary and developmental optimization processes, as well as prior learning, on the learning capability of brain networks. We use the terms training and optimization interchangeably. Like in Hochreiter et al., 2001 we let all synaptic weights of the network belong to the set of hyper-parameters that are optimized through the outer loop. Hence the outer loop training shapes the activation dynamics of the network, which include its firing activity and short-term memory. The synaptic weights of the neural network can encode an efficient algorithm for learning arbitrary tasks from the family of tasks.

When the brain learns to predict sensory inputs, or state changes that result from an action, this can be formalized as learning from a teacher (i.e., supervised learning). The teacher is in this case the environment, which provides – often with some delay – the target output of a network. Here, we show that LSNNs can learn using this supervision signal without modifying their synaptic weights.

4.3 L2L allows installing prior knowledge into networks of spiking neurons

We first demonstrate the principles of the new methods for fast learning in SNNs for a rather simple but transparent example: Learning a specific input-output mapping from a parametrized range of such mappings. More concretely, we consider the task of learning a class of sinusoidal functions using supervision. The family of tasks $\mathcal{F}$ was defined as consisting of sinusoidal functions with different phases and amplitudes, parameterized as $y = Asin(x + \phi)$, where x was in the range of

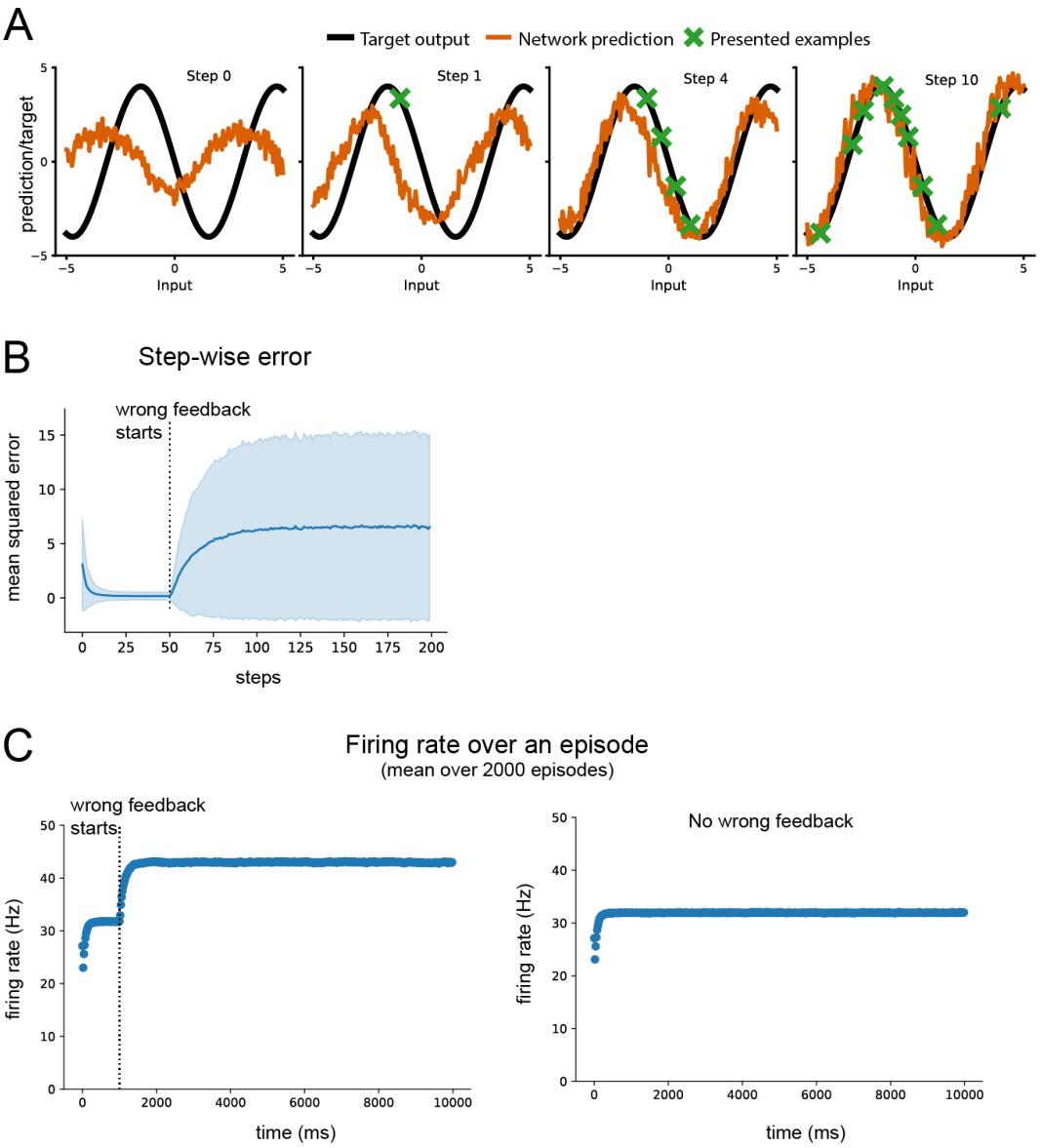

Figure 4.2: Prior learning and effect of feedback disruption. (**A**) Illustration of the prior knowledge acquired by the LSNN through L2L for another family $\mathcal{F}$ (sinus functions). Even adversarially chosen examples (Step 4) do not induce the LSNN to forget its prior. (**B**) Step-wise error before and after the feedback is switched to wrong feedback. (**C**) Firing rate of the network increases after the switch to wrong feedback (left), whereas it remains at a lower value as long as the feedback is correct (right). Shown firing rate is mean over 2000 episodes.

$[-5, 5]$, and A and ϕ were chosen uniformly randomly between $[0, \pi]$ and $[0.1, 5]$ respectively (see Fig. 4.2A).

During training, both the phase and amplitude were randomly chosen anew for each episode. In each step k of an episode, the network was given, as input, a value of x^k chosen randomly from the above range, and was expected to produce, as output, a prediction $C(x)$ that matched the target $y^k = Asin(x^k + \phi)$ for the values of A and ϕ chosen in that particular episode, where k denotes the step index. The network was also given the correct target of the previous step i.e. with a delay of one step as in Hochreiter et al., 2001, and only after the network had guessed the output value for the preceding input. For the outer-loop, the LSNN parameters were updated using the version of BPTT for spiking neurons Bellec et al., 2018b, to minimize the squared loss of the prediction error. Updates to the network weights were done after each batch of 20 episodes, which formed one iteration of the outer-loop. Since each individual episode in the batch used a sinusoidal curve with different parameters, the network could not just learn to predict values for one particular sinusoidal curve, but had to learn to learn the parameters of each different sinusoid in each episode.

After training, the weights of the network remained fixed, and the network was just given inputs and feedback as described above. It was required to learn to make correct predictions of sinusoids with new parameters in an online manner, using just its short-term memory and dynamics.

After a few thousand training iterations in the outer loop, the LSNN achieved low mean-squared error for learning new sinusoidal functions from the family $\mathcal{F}$. In fact, the LSNN was able to produce good approximations of a new sinusoidal function in just a few steps. Each sequence of examples evokes an internal model that is stored in the short-term memory of the LSNN. Fig. 4.2A shows the fast evolution of internal models of the LSNN during the first few steps. We make the current internal model of the LSNN visible by probing its prediction $C(x)$ for hypothetical new inputs x in the domain (without allowing it to modify its short-term memory; all other inputs advance the network state according to the dynamics of the LSNN). It is evident that the internal model of the LSNN is a smooth sinusoid function from the beginning. Within a few trials this smooth function approximated the currently chosen sinusoid quite well. Hence, during the outer-loop training of L2L, the LSNN had acquired a prior for the types of functions that are to be learnt, that was encoded in its synaptic weights. These results suggest that L2L is able to install some form of prior knowledge about the task in the LSNN.

This result indicates that the LSNN fits internal models for smooth functions to the examples it received. Even when we selected examples in an adversarial manner, which happened to be in a straight line (as in the presented examples in Fig. 4.2A), this did not disturb the prior knowledge of the LSNN. Altogether the network learning that was induced through L2L in the LSNNs is of particular interest from the perspective of the design of learning algorithms: We are not aware of

previously documented methods for installing structural priors for online learning of a recurrent network of spiking neurons. Additionally, the ability to install such priors would be essential for ensuring safety and robustness in SNNs when used in real world applications.

We then tested the effect of switching to giving wrong feedback to the network after it had learnt a particular task. As usual, within an episode, the specific sinus task parameters were fixed. The network predicted the values of the function with high accuracy by the 50th step in almost all episodes. We then replaced the target feedback with a random signal after the 50th step, and observed the change in neural activity in the network. As expected, the network unlearned the particular function it had learnt and the error increased significantly as shown in Fig. 4.2B. The error stayed high after step 50 since the feedback received by the network was random, and the network was unable to learn any meaningful function using this feedback. Interestingly, we also observed that the firing rate of the network increased significantly when we started giving it the wrong feedback, and stayed high as long as we continued to give wrong feedback — this is shown in Fig. 4.2C. We show the mean values over 2000 such episodes in Fig. 4.2B,C which validates that this phenomena is robust to the specific sinusoidal task chosen in every episode. This increase in firing activity could provide an experimentally testable signature of meta-learning in the brain if the overall task setup is matched.

4.4 L2L supports fast learning of motor prediction by networks of spiking neurons

We next turn to a more biologically realistic task that is also harder — learning a forward model of the dynamical system formed by our body and muscles. The brain needs this to be able to plan movement and also to take corrective action if the desired motor state is not achieved Lalazar and Vaadia, 2008; Wolpert and Ghahramani, 2000. Therefore, motor prediction plays an important role in controlling movement of the body and more generally behaviour. Visual and proprioceptive feedback from voluntary movement Wong et al., 2012 enables the brain to learn how the body responds to various motor commands.

But one important question that has remained in neuroscience is how neurons rapidly adapt their activity to continue making correct predictions when the kinematic and dynamic properties of the body undergoes a change. For example, carrying objects can change the effective mass of the limbs, and using tools can change the effective lengths of the limbs. And yet, the brain is able to quickly correct for these changes without requiring multiple rounds of trial and error over longer periods of time.

Learning over days or weeks is usually associated with persistent synaptic changes Kleim et al., 2004, but behaviour can adapt a lot more rapidly Thoroughman and Shadmehr, 2000, sometimes also in a single trial Bailey and Chen, 1988. Recent

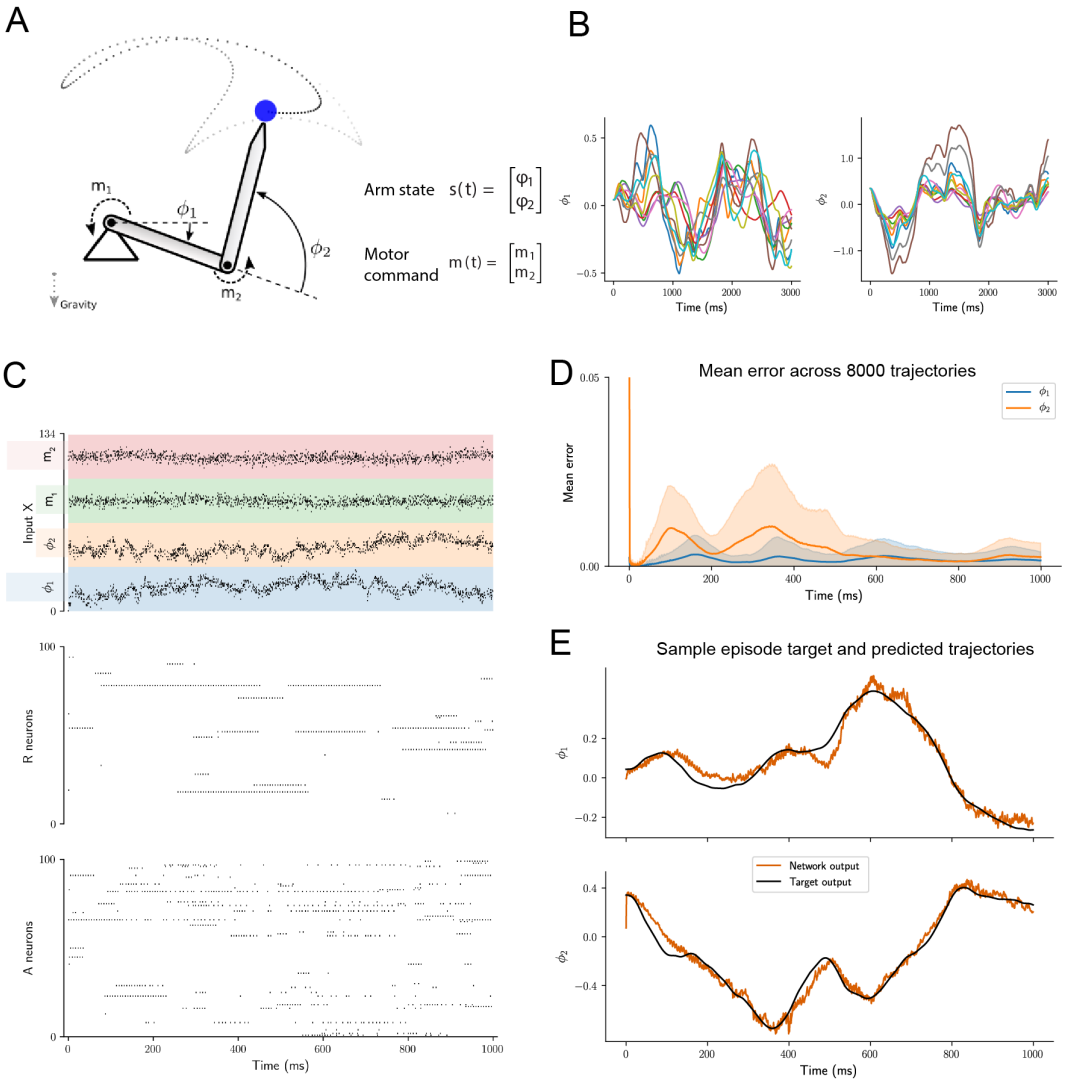

Figure 4.3: Learning to learn forward models for motor control from few trials. (**A**) Illustration of the two-link arm model with states given by the angle of the links, and the motor command applied on both the joints. (**B**) Sample trajectories generated by the two-link arm with different link masses and lengths. (**C**) Spike raster of the LSNN for 1 second of an inner-loop learning episode (after outer-loop training). (**D**) Mean error ever all test episodes during the 1 second of inner-loop learning. (**E**) Target trajectories and network prediction for one sample test episode for an arm with new link lengths and masses.

work in neuroscience Perich et al., 2018 indicates that this rapid adaptation of motor output occurs even while the functional connectivity in cortical areas remains unaltered. In fact, it is quite implausible that the strengths of synaptic connections between neurons in our brain are changed when we, say, take a book into a hand.

We wondered if learning to learn implemented in SNNs with fixed weights could provide a basis for explaining how motor prediction can be rapidly adapted/learnt without synaptic changes. Such a model could be very powerful when applied to robotics where such rapid adaptation is not only useful, but in many cases, absolutely essential. The ability to port this model onto neuromorphic hardware, which would then allow the model to run with high performance and power efficiency, also makes it a good match for robotics applications.

Therefore, we now explore how learning to learn (L2L) can generate recurrent networks of spiking neurons that are able to exhibit rapid learning of motor prediction without synaptic plasticity, even for kinematic and dynamic configurations that have not been encountered before.

We considered the task of predicting the state of a two-link arm model as illustrated in Fig. 4.3A. One link is connected by a joint to the other link, which is itself connected by a joint to a fixed position in space. Torque could be applied to each of the two joints, which results in movement. Both links are also subject to gravity at all times. Concretely, the task that we considered here was to predict the angles of the two joints. But the masses and lengths of the two links were different in every episode, which defined the family of tasks $\mathcal{F}$. This led to very different trajectories when the torque was applied — Fig. 4.3B shows examples of state trajectories that result for arms with different link masses and lengths. We provided feedback to the network so that it could learn about the mass and length configuration of the current task. This feedback was in the form of the true angles of the links, but delayed by 100 ms.

For each episode in the inner-loop, a new arm was chosen with different masses and lengths of links, and the LSNN was asked to predict the sequence of states of the arm given the torques. In the outer-loop, we trained the LSNN using BPTT for spiking neurons, to minimize the squared error between the predictions and the target arm states. (See Chapter D for further details).

After outer-loop training, we fixed the weights of the LSNN and its readouts, and used it to predict the angles of the two joints of an arm with previously unseen link lengths and masses. Fig. 4.3E shows a sample episode of 1 second during the inner loop learning process with fixed synaptic weights. It is evident that the network was able to adapt its predictions to the new lengths and masses fairly quickly. The error in prediction, averaged over 8000 such episodes, is shown in Fig. 4.3D. This demonstrates that the network achieves low prediction error within about $800 - 1000$ ms on average. This is in spite of the fact that it has not already been trained on arms with these specific link lengths and masses (since these are chosen randomly for each episode). Note that the drop in the error in the first 100 ms is actually caused by the fact that the arm always starts from the same initial

position, but the effect of the different link lengths and mass on the arm trajectory is observed only after the first 100 ms. The inner-loop learning happens significantly faster than earlier biologically plausible learning algorithms with spiking neurons, for example Gilra and Gerstner, 2017.

4.5 Meta-RL enables fast learning of complex navigation tasks

We now turn to an application of meta reinforcement learning (meta-RL) to LSNNs. In meta-RL, the LSNN receives rewards instead of teacher inputs. Meta-RL has led to a number of remarkable results for LSTM networks, see e.g. J. X. Wang et al., 2016; Duan et al., 2016. In addition, J. X. Wang et al., 2018 demonstrates that meta-RL provides a very interesting perspective of reward-based learning in the brain.

We focused on one of the more challenging demos of J. X. Wang et al., 2016 and Duan et al., 2016, where an agent had to learn to find a target in a 2D arena, and to navigate subsequently to this target from random positions in the arena. This task is related to the well-known biological learning paradigm of the Morris water-maze task Morris, 1984; Vasilaki et al., 2009. We study here the capability of an agent to discover two pieces of abstract knowledge from the concrete setup of the task: The distribution of goal positions, and the fact that the goal position is constant within each episode. We asked whether the agent would be able to exploit the pieces of abstract knowledge from learning for many concrete episodes, and use it to navigate more efficiently.

An LSNN-based agent was trained on a family of navigation tasks with continuous state and action spaces in a circular arena. The task is structured as a sequence of episodes, each lasting 2 seconds. The goal was placed randomly for each episode on the border of the arena. When the agent reached the goal, it received a reward of $+1$, and was placed back randomly in the arena. When the agent hit a wall, it received a negative reward of -0.02 and the velocity vector was truncated to remain inside the arena. The objective was to maximize the number of goals reached within the episode. This family $\mathcal{F}$ of tasks is defined by the infinite set of possible goal positions. For each episode, an optimal agent is expected to explore until it finds the goal position, memorize it and exploits this knowledge until the end of the episode by taking the shortest path to the goal. We trained an LSNN so that the network could control the agent's behaviour in all tasks, without changing its network weights.

Since LSNNs with just a few hundred neurons are not able to process visual input, we provided the current position of the agent within the arena through a place-cell-like Gaussian population-rate encoding of the current position. The lack of visual input already made it challenging to move along a smooth path, or to stay within a safe distance from the wall. The agent received information

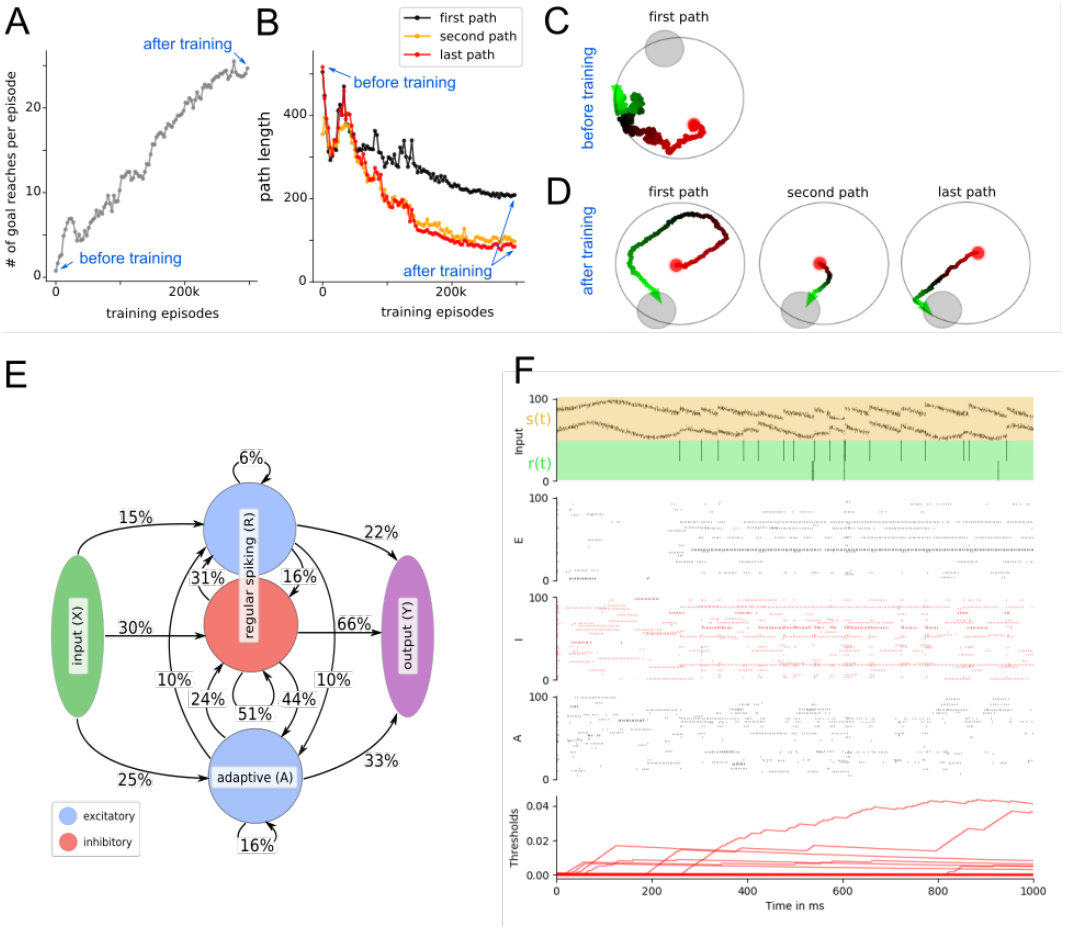

Figure 4.4: Meta-RL results for an LSNN. (**A, B**) Performance improvement during training in the outer loop. (**C, D**) Samples of navigation paths produced by the LSNN before and after this training. Before training, the agent performs a random walk (**C**). In this example it does not find the goal within the limited episode duration. After training (**D**), the LSNN had acquired an efficient exploration strategy that uses two pieces of abstract knowledge: that the goal always lies on the border, and that the goal position is the same throughout an episode. Note that all synaptic weights of the LSNNs remained fixed after training. (**E**) Connectivity between sub-populations of the network after training. The global connectivity in the network was constrained to 20%. (**F**) The network dynamics that produced the behavior shown in (**A**).

about positive and negative rewards in the form of spikes from external neurons. For training in the outer loop, we used BPTT together with DEEP R Bellec et al., 2018a applied to the surrogate objective of the Proximal Policy Optimization (PPO) algorithm Schulman et al., 2017. In this task the LSNN had 400 recurrent units (200 excitatory, 80 inhibitory and 120 adaptive neurons with adaptation time constant τ_a of 1200ms), the network was rewired with a fixed connectivity of 20%. The resulting network diagram and spike raster is shown in Fig. 4.4E,F.

The network behaviour before, during, and after L2L optimization is shown in Fig. 4.4C,D. Fig. 4.4A shows that a large number of training episodes finally provided significant improvements. With a close look at Fig. 4.4B, one sees that before 52k training episodes, the intermediate path planning strategies did not seem to use the discovered goal position to make subsequent paths shorter. Hence the agents had not yet discovered that the goal position does not change during an episode. After training for 300k episodes, one sees from the sample paths in Fig. 4.4D that both pieces of abstract knowledge had been discovered by the agent. The first path in Fig. 4.4D shows that the agent exploits the fact that the goal is located on the border of the maze. The second and last paths show that the agent knows that the position is fixed throughout an episode. Altogether this demo shows that meta-RL can be applied to RSNNs, and produces previously unseen capabilities of sparsely firing RSNNs to extract abstract knowledge from experimentation, and the ability to use it in clever ways for controlling behaviour. The trained LSNN is capable of autonomously controlling a complex multi-stage behaviour consisting of exploration and exploitation, which has not been demonstrated before in networks of spiking neurons.

4.6 Discussion

We know that fast learning and rapid adaptation is fairly common in biology at time scales that are not explained by the currently studied plasticity mechanisms. Here, we propose a mechanism to achieve this rapid learning in a very general and biologically plausible way — using only the dynamics of a recurrent spiking network with few constraints on neuron or plasticity mechanisms. This mechanism uses learning-to-learn to install and enable this form of fast learning for families of tasks, including motor prediction and navigation. We consider processes like evolution and development as potential mechanisms for outer-loop optimization to have imbued the LSNN with learning potential in the inner-loop, and we emulate it with BPTT.

This L2L mechanism also provides a powerful way to install prior knowledge about the structure of the family of tasks into the LSNN. We demonstrate it with a simple structured task in Fig. 4.2A, where, once the network has learnt the overall task structure, it is able to robustly ignore other misleading and more local structures. We also show that giving wrong feedback to the network after it has learnt to predict values for a single task leads to elevated neural activity as seen in Fig. 4.2B.

This provides an experimentally testable hypothesis, and provides an alternative explanation for elevated activity resulting from unexpected or wrong feedback.

We then show that we can use this framework to generate motor prediction models that are able to adapt very quickly to changes in the dynamics of the plant being controlled. In our particular case, the arm masses and lengths changed leading to the change in dynamics in a two-dimensional two-link arm model. We speculate that this could be further extended to provide a biologically plausible mechanism that can explain rapid adaptation in motor control without changes in synaptic plasticity as observed in Perich et al., 2018.

Finally, we show that this framework can solve the Morris water-maze task, which is a well known biological learning paradigm Morris, 1984; Vasilaki et al., 2009. The task we considered was a continuous control problem that we solved with meta-reinforcement learning, where the goal position changed after every episode consisting of many trials. Learning to learn enables the LSNN — without any additional outer control or clock — to embody an agent that first searches an arena for a goal, and subsequently exploits the learnt knowledge in order to navigate fast from random initial positions to this goal.

Altogether, we expect that the new methods and ideas that we have introduced will advance our understanding and reverse engineering of RSNNs in the brain. For example, the RSNNs that emerged in all the tasks compute and learn with a brain-like sparse firing activity, quite different from an SNN that operates with rate-codes. Apart from these implications for computational neuroscience, our finding that RSNNs can acquire powerful computing and learning capabilities with very energy-efficient sparse firing activity provides new application paradigms for spike-based computing hardware.

Chapter 5

Slow processes of neurons enable a biologically plausible approximation to policy gradient

Contents

Abstract. Recurrent neural networks underlie the astounding information processing capabilities of the brain, and play a key role in many state-of-the-art algorithms in deep reinforcement learning. But it has remained an open question how such networks could learn from rewards in a biologically plausible manner, with synaptic plasticity that is both local and online. We describe such an algorithm that approximates actor-critic policy gradient in recurrent neural networks. Building on an approximation of backpropagation through time (*BPTT*): *e-prop*, and using the equivalence between forward and backward view in reinforcement learning (RL), we formulate a novel learning rule for RL that is both online and local, called *reward-based e-prop*. This learning rule uses neuroscience-inspired slow processes and top-down signals, while still being rigorously derived as an approximation to actor-critic policy gradient. To empirically evaluate this algorithm, we consider a delayed reaching task, where an arm is controlled using a recurrent network of spiking neurons. In this task, we show that *reward-based e-prop* performs as well as an agent trained with actor-critic policy gradient with biologically implausible *BPTT*.

Acknowledgments and author contributions. This chapter is based on the manuscripts

Anand Subramoney*, Franz Scherr*, Guillaume Bellec*, Elias Hajek, Darjan Salaj, Robert Legenstein, Wolfgang Maass (2019). "Slow processes of neurons enable a biologically plausible approximation to policy gradient." *NeurIPS 2019 Workshop: Biological and artificial Reinforcement Learning*.

To this study, I contributed as first author along with FS, GB. The study was conceived by WM, AS, GB, FS, with the theory being developed by AS, FS, GB. The experiments were designed by WM, AS, GB and were conducted by AS. The manuscript was written by AS, FS, GB, EH, DS, RL, WM.

5.1 Introduction

Deep reinforcement learning (deep RL) has formed the basis for the staggeringly successful recent results in machine learning and AI Mnih et al., 2015b; Vinyals et al., 2019; Silver et al., 2018. A standard algorithm in deep RL is actor-critic policy gradient Mnih et al., 2016, where the network model outputs probabilities for each action (called the policy) and also predicts the sum of future rewards (called the value). For tasks that require working memory, recurrent neural networks (RNNs) are used, often in the form of LSTM units Hochreiter and Schmidhuber, 1997 in deep RL. These recurrent networks are trained using *backpropagation through time* (*BPTT*) to maximize the expected sum of future rewards while minimizing the error in predicting the value.

But *BPTT* requires storing the intermediate states of all neurons during a network computation, and merging these in a subsequent offline process with gradients that are computed backwards in time. This makes it very unlikely that *BPTT* is used by the brain Lillicrap and Santoro, 2019. In addition, assigning credit to actions necessitates simulating the environment until the outcome becomes evident, i.e. until the end of an episode. This makes online implementation of actor-critic policy gradient difficult.

In this submission, we describe an algorithm called *reward-based e-prop* that solves both of these problems. The result is a local, online RL rule that can be used to train RNNs. It can also be shown to approximate the ideal learning rules based on gradient descent. This algorithm builds on an approximation to *BPTT* called *e-prop* Bellec et al., 2019b and incorporates the advantages of the actor-critic method into a learning rule that is both local and online.

This method derives its inspiration from experimental data in biology for how such a learning rule could be implemented in a biologically plausible way. The dynamics of neurons are known to have traces of past activity on a molecular level Sanhueza and Lisman, 2013. These traces record events that are known to induce plasticity in the presence of top-down learning signals Cassenaer and Laurent, 2012; Yagishita et al., 2014; Gerstner et al., 2018. These type of local traces are referred to as eligibility traces in *e-prop* Bellec et al., 2019b. In addition, various kinds of top-down signals that are specific for target population of neurons are known to exist MacLean et al., 2015. Among them are signals that predict upcoming rewards Engelhard et al., 2019; Roeper, 2013 or movement errors in case of error-related negativity (ERN) Sajad et al., 2019. These signals constitute

the learning signals of *e-prop* and the reward prediction error that emerges in the context of *reward-based e-prop*.

Using such processes, and formulating online actor-critic policy gradient R. S. Sutton and Barto, 2018 using the mathematical framework of *e-prop* leads us to *reward-based e-prop*. In *reward-based e-prop*, the weight updates can be calculated at every time step without having to wait for all future rewards, without backpropagation of signals either through time (online) or synapses (local). We demonstrate this learning rule on a neuroscience inspired RL task (Fig. 2.1). We specifically consider the scenario where the agent needs to remember early parts of an episode in order to produce a successful policy in later parts of an episode. This requires the use of RNNs for working memory, and the ability to discover the relationship between early observations and later actions.

5.2 Reward-based learning

In the RL setting an agent interacts with an environment. In our case, the agent is implemented by a recurrent network, specifically a recurrent network of spiking neurons with working memory (see Appendix for details), although the theory is generally applicable to any model of recurrent network. This network receives observations $\boldsymbol{x}^t$ from the environment, and interacts with it by means of real-valued actions $\boldsymbol{a}^t$. These actions are distributed according to a Gaussian centered around the network output $\boldsymbol{y}^t$ with a fixed variance σ^2. The probability distribution of these actions $\pi(\boldsymbol{a}^t|\boldsymbol{y}^t)$ is the stochastic policy. In our setting, we restrict the actions to certain decision times $t_0, \ldots, t_n, \ldots$, and set $\pi \equiv 0$ at other times. The environment can provide a positive or negative reward r^t at any time t to inform the agent about favorable states.

The goal of reward-based learning is to maximize sum of discounted future rewards in expectation: $\max \mathbb{E}[R^0]$, where we define the return at time t as $R^t = \sum_{t' \geq t} \gamma^{t'-t} r^{t'}$. Here, $\gamma \leq 1$ is known as the discounting factor. This is achieved by the actor-critic variant of policy gradient, involving the policy π (the actor) and an additional output neuron V^t (with weights $W_j^{V,\text{out}}$), predicting the value function $\mathbb{E}[R^t]$ (the critic). Both are learnt simultaneously by minimizing the loss function:

$$E = E_\pi + c_V E_V, \tag{5.1}$$

where $E_\pi = -\mathbb{E}[R^0]$ measures the performance of the stochastic policy π, $E_V = \mathbb{E}[\sum_t \frac{1}{2}(R^t - V^t)^2]$ measures the error in value prediction, and c_V is a hyperparameter chosen appropriately.

To improve the agent's behavior, we minimize the loss function E using gradient descent on the network weights W_{ji}. For this purpose, we can derive an estimator $\widehat{\frac{dE}{dW_{ji}}}$

of the loss gradient with reduced variance (using the policy gradient theorem R. S. Sutton and Barto, 2018):

$$\frac{dE}{dW_{ji}} = \mathbb{E}\Bigg[\underbrace{-\sum_t (R^t - V^t)\left(\frac{d}{dW_{ji}}\big(\log \pi(\boldsymbol{a}^t|\boldsymbol{y}^t) + c_V V^t\big)\right)}_{\stackrel{\text{def}}{=} \widehat{\frac{dE}{dW_{ji}}}}\Bigg] . \tag{5.2}$$

Equation (5.2) can be seen as the typical formulation of an actor-critic algorithm. However, in its current form, the value of this gradient cannot be computed in a biological plausible way:

A) the quantity R^t is not available at time t because its computation requires future rewards,
B) calculating the derivatives of the network output ($\log \pi$, V) with respect to the network input and recurrent weights requires biologically implausible *BPTT*.

Solution to A): Backward view We can alleviate the first problem involving the computation of R^t at time t by exploiting the equivalence between forward view and backward view in RL R. S. Sutton and Barto, 2018. For this purpose, we introduce a temporal difference (TD) error $\delta^t = r^t + \gamma V^{t+1} - V^t$ that allows us to rewrite the error in value prediction by a sum over future TD errors: $R^t - V^t = \sum_{t' \geq t} \gamma^{t'-t} \delta^{t'}$. If we substitute this into the relevant part of equation (5.2), we obtain two sums over t and t'. Because the summation of t' starts from t, we can reorganize and collect terms such that only past terms are summed (commonly denoted backward view of RL R. S. Sutton and Barto, 2018):

$$\sum_t (R^t - V^t)(\,\cdot\,) = \sum_t \sum_{t' \geq t} \gamma^{t'-t} \delta^{t'}(\,\cdot\,) = \sum_{t'} \delta^{t'} \sum_{t \leq t'} \gamma^{t'-t}(\,\cdot\,) = \sum_{t'} \delta^{t'} F_\gamma(\,\cdot\,) . \tag{5.3}$$

In order to simplify notation, we introduce a temporal filter $\mathcal{F}$ that is defined as $\mathcal{F}_\alpha(x^t) = \alpha \mathcal{F}_\alpha(x^{t-1}) + x^t$.

Solution to B): *E-prop* Computing the gradient $\frac{d}{dW_{ji}}\big(\log \pi(\boldsymbol{a}^t|\boldsymbol{y}^t) + c_V V^t\big)$ required in equation (5.2) with respect to input weights W_{ji}^{in} and recurrent weights W_{ji}^{rec} in RNNs with *BPTT* is biologically implausible. We approximate this gradient using *e-prop* Bellec et al., 2018b by ignoring both spatially and temporally non-local interactions. This gradient is approximated as a product of learning signals L_j^t, which are specific to the post-synaptic neuron j, and synapse-specific eligibility traces e_{ji}^t:

$$\widehat{\frac{d}{dW_{ji}}}\big(\log \pi(\boldsymbol{a}^t|\boldsymbol{y}^t) + c_V V^t\big) = L_j^t \bar{e}_{ji}^t , \tag{5.4}$$

Here, we define $\bar{e}_{ji}^t = \mathcal{F}_\kappa(e_{ji}^t)$ with κ being the time constant associated with the output neuron, and the eligibility traces e_{ji}^t only depend on the activity that is local to the synapse. The eligibility traces are formally defined as $e_{ji}^t = \frac{\partial z_j^t}{\partial \boldsymbol{h}_j^t} \cdot \boldsymbol{\epsilon}_{ji}^t$, using a so-called eligibility vector that is propagated forward in time, according to the dynamics of neurons: $\boldsymbol{\epsilon}_{ji}^{t+1} = \frac{\partial \boldsymbol{h}_j^{t+1}}{\partial \boldsymbol{h}_j^t} \cdot \boldsymbol{\epsilon}_{ji}^t + \frac{\partial \boldsymbol{h}_j^{t+1}}{\partial W_{ji}}$, where $\boldsymbol{h}_j^t$ is the hidden state vector and z_j^t is the observable state of the neuron model. This definition precisely captures the dynamics of the hidden states. In particular, if the dynamics of the neurons contain slower processes as our network does (see Appendix), long lived traces arise that can help with the credit assignment problem.

The learning signal $L_j^t = \frac{\partial\left(\log \pi(\boldsymbol{a}^t|\boldsymbol{y}^t) + c_V V^t\right)}{\partial z_j^t}$ is the impact of neuron spikes z_j^t on $\log \pi(\boldsymbol{a}^t|\boldsymbol{y}^t) + c_V V^t$. Considering our Gaussian policy π, we obtain:

$$L_j^t = -c_V W_j^{V,\text{out}} + \sum_k W_{jk}^{\pi,\text{out}} \frac{y_k^t - a_k^t}{\sigma^2} \ . \tag{5.5}$$

where $W_{jk}^{\pi,\text{out}}$ and $W_j^{V,\text{out}}$ are the output weights for the policy π and the value prediction V respectively.

5.3 Learning rule emerging from *reward-based e-prop*

The learning rule emerging in equation (5.5) solves major issues of biological plausibility in *BPTT*. There is no need to backpropagate through time or store earlier network activity, and the algorithm can be implemented by online updates. Even the sum over t in (5.5) does not need store every element of the sum. Instead the sum can be accumulated, or every contribution can be applied directly.

With this definition of the learning signal (equation (5.5)), the deviation between the stochastic action a_k^t and its expected value y_k^t are fed back to neuron j with the same weights $W_{kj}^{\pi,\text{out}}$ that define y_k^t, and this symmetry also arises for $W_j^{V,\text{out}}$. To avoid such biologically implausible weight sharing between the feedforward and the feedback pathways, we use fixed random feedback weights B_{jk}^π, B_j^V as in (Lillicrap et al., 2016; Nøkland, 2016). Using a simple local plasticity rule on B_{jk} that mirrors the plasticity rule applied to W_{kj} also leads to similar results. Putting together the results of equation (5.3), (5.4) and (5.5), we obtain the learning rule for *reward-based e-prop*:

$$\Delta W_{ji}^{\text{rec}} = -\eta \sum_t \delta^t \mathcal{F}_\gamma\left(L_j^t\, \bar{e}_{ji}^t\right) \quad \text{for} \tag{5.6}$$

$$L_j^t = -c_V B_j^V + \sum_k B_{jk}^\pi \frac{y_k^t - a_k^t}{\sigma^2} \ , \tag{5.7}$$

where η is a fixed learning rate.

Note that the filtering $\mathcal{F}_\gamma$ requires an additional eligibility trace per synapse. The term $\mathcal{F}_\gamma\left(L_j^t\ \bar{e}_{ji}^t\right)$ is identical to the eligibility traces commonly used in RL R. S. Sutton and Barto, 2018; Seijen and R. Sutton, 2014, except that we use an approximation for the true gradient of the loss instead. Additionally, Seijen and R. Sutton, 2014 use a linear feedforward model, whereas we specifically address learning in recurrent neural networks. This term depends on the learning signal and does not have the same function as the eligibility trace e_{ji}^t that are employed by *e-prop*.

5.4 Empirical evaluation of *reward-based e-prop*

We tested *reward-based e-prop* on a task that is representative of a common learning experiment paradigm in systems neuroscience: There an agent has to learn a delayed goal-directed movement, consisting of a sequence of many 2-dimensional continuous motor commands. The rewards are sparse and often arrive long after relevant actions have been taken. The agent first receives a spatial goal cue (Fig. 5.1C), which is followed by a delay period during which the agent has to remember the cue, and has to make sure the tip of the two-joint arm it controls remains within a center region (indicated by a dotted circle) in order to avoid punishment. The agent controls the arm movement by controlling the angular velocities of the two joints. After the delay period, the agent receives a go-cue, which instructs that the agent has to move the tip of the arm to the location that was indicated by the initial goal cue in order to receive a positive reward.

Note that the network was not given any forward- or inverse model of the arm but had to learn those implicitly. The agent also required working memory to remember the goal cue from the beginning of the episode until it received the go-cue. In addition, due to delayed rewards, credit assignment for actions which led to the reward was non-trivial.

We used a recurrent spiking network with an additional working memory mechanism called an LSNN (as in Bellec et al., 2018b) to control the arm. The overall architecture of the network, along with the components that contribute to the weight update rule, are shown in Fig. 5.1A.

Three sample trials after learning are shown in Fig. 5.1D. Fig. 5.1B shows that *reward-based e-prop* is able to solve this demanding RL task about as well as policy gradient with biologically implausible *BPTT*. This is due to the fact that the eligibility traces that arise in *reward-based e-prop* are able to handle the long term credit assignment problem, while the slower dynamics of the neurons in the network provide the working memory. Note that the credit assignment here occurs successfully without any backpropagation of gradients through the entire history. We conjecture that

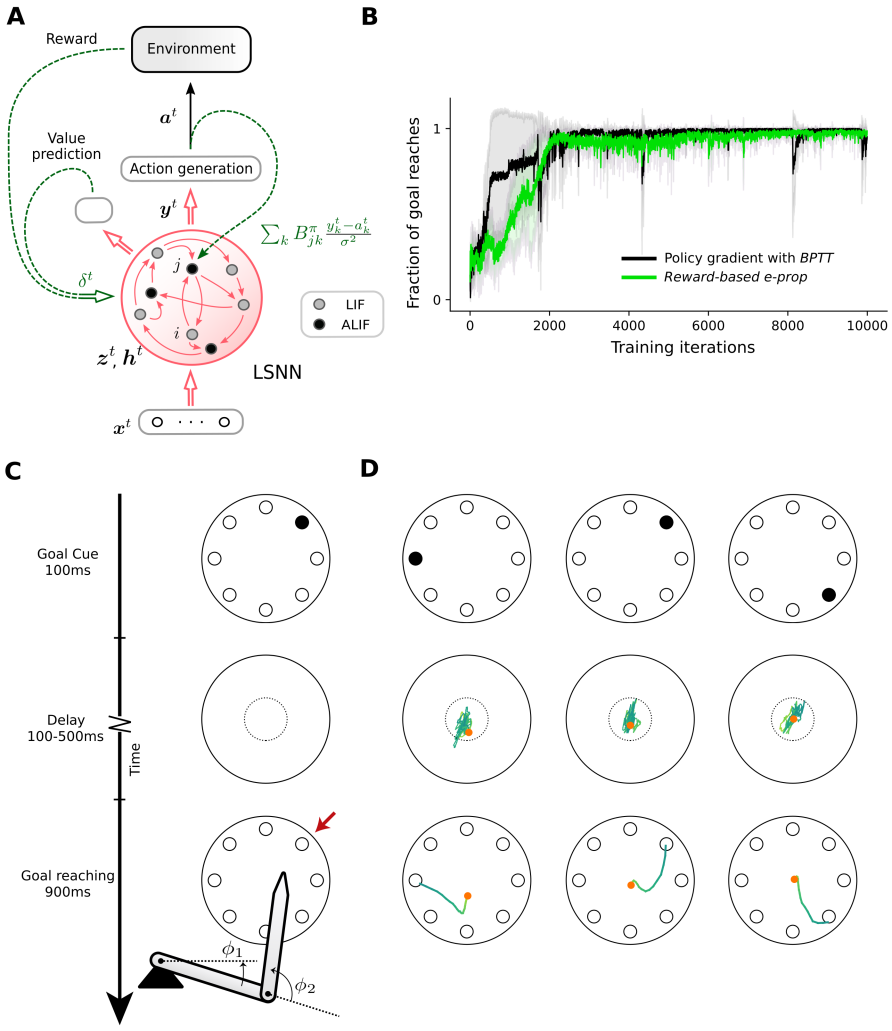

Figure 5.1: Application of *e-prop* to RL. A) Learning architecture for *reward-based e-prop*: The network input $\mathbf{x}^t$ consists of the current joint angles and input cues. The network produces output $\mathbf{y}^t$ which is used to stochastically generate the actions $\mathbf{a}^t$. In addition, the network produces the value prediction, which, along with the reward from the environment, is used to calculate the TD-error δ^t, An LSNN is the network model defined in the Appendix. The learning signals and the TD-errors are used to calculate the weight update, as denoted by the green dotted lines. **B**) Performance of *reward-based e-prop* and of a control where *e-prop* is replaced by *BPTT*, both for an LSNN consisting of 350 LIF and 150 adaptive LIF (ALIF) neurons. Solid curves show the mean over 5 different runs, and shaded area indicates 1 standard deviation. **C**) Scheme of the delayed arm movement task. The red arrow points to the formerly visible goal. The arm always starts moving from the center of the circle. **D**) Resulting arm movement in three sample trials after learning. The orange dot indicates the position of the tip of the arm at the end of the delay period.

variants of *reward-based e-prop* will be able to solve most RL tasks that can be solved by online actor-critic methods in machine learning.

5.5 Discussion

We have proposed a biologically plausible learning rule – *reward-based e-prop* – for RL that is both online and local. The combination of reward prediction error and neuron-specific learning signal has also been used in a plasticity rule for feedforward networks inspired by neuroscience Pozzi et al., 2018. But in the case of *reward-based e-prop*, it arises from the approximation of *BPTT* by *e-prop* in spiking RNNs tackling RL problems. Other previously proposed learning rules use the correlation of the noisy output of network neurons with rewards to estimate the gradients of the policy Kappel et al., 2018; Gerstner et al., 2018. But due to noisy gradient estimates, such learning rules are inefficient.

Since *reward-based e-prop* is based on a transparent mathematical principle, it provides a normative model for signals that are abundantly present in the brain, but whose precise role is not very well understood – eligibility traces, learning signals and reward prediction errors. Actor-critic policy gradient combined with BPTT for RNNs has been shown to be very powerful in deep RL. *Reward-based e-prop* approximates this, and so has the potential to be a very powerful online learning algorithm for RL for a wide variety of tasks. Here, we have demonstrated that it can be used to successfully train recurrent networks on a reinforcement learning task that requires complex arm movements, working memory and long term credit assignment. In addition, we have demonstrated this using a recurrent network model that is biologically realistic – using spiking neurons, and slower processes very similar to that found in biology.

Future work would explore scaling up this algorithm to tasks of greater complexity and memory demands. Furthermore, an online RL algorithm such as *reward-based e-prop* can also lead to a qualitative jump in on-chip learning capabilities of neuromorphic chips.

Appendix

Appendix A

List of publications

1. Anand Subramoney, Guillaume Bellec, Franz Scherr, Robert Legenstein, Wolfgang Maass (2020). "Prior knowledge and network dynamics enable networks of spiking neurons to learn new tasks from just a few trials." *(In preparation)*.
2. Anand Subramoney, Arjun Rao, Robert Legenstein, Wolfgang Maass (2020). "Dendritic learning in layer 5 pyramidal cells approximates logistic regression." *(In preparation)*.
3. Anand Subramoney, Wolfgang Maass (2020). "The critical regime enables networks of spiking neurons to maintain a working memory during cognitive computations." *(In preparation)*.
4. Darjan Salaj*, Anand Subramoney*, Ceca Kraisnikovic*, Guillaume Bellec, Robert Legenstein, Wolfgang Maass (2020). "Spike-frequency adaptation provides a long short-term memory to networks of spiking neurons." *(submitted for publication. biorXiv:10.1101/2020.05.11.081513)*.
5. Anand Subramoney, Franz Scherr, Wolfgang Maass (2019). "Reservoirs learn to learn." *(submitted for publication. arXiv:1909.07486)*.
6. Anand Subramoney*, Guillaume Bellec*, Franz Scherr*, Elias Hajek, Darjan Salaj, Robert Legenstein, Wolfgang Maass (2019). "Slow processes of neurons enable a biologically plausible approximation to policy gradient." *NeurIPS 2019 Workshop: Biological and artificial Reinforcement Learning, Advances in Neural Information Processing Systems 32*.
7. Guillaume Bellec*, Franz Scherr*, Elias Hajek, Darjan Salaj, Anand Subramoney, Robert Legenstein, Wolfgang Maass (2019). "Eligibility traces provide a data-inspired alternative to backpropagation through time." *NeurIPS 2019 Workshop: Real neurons and hidden units, Advances in Neural Information Processing Systems 32*.
8. Guillaume Bellec*, Franz Scherr*, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, Wolfgang Maass (2019). "A solution to the learning dilemma for recurrent networks of spiking neurons." *(submitted for publication. biorXiv:10.1101/738385)*.
9. Jacques Kaiser*, Michael Hoff*, Andreas Konle, Juan Camilo Vasquez Tieck, David Kappel, Daniel Reichard, Anand Subramoney, Robert Legenstein, Arne Roennau, Wolfgang Maass, Rüdiger Dillmann (2019).

"Embodied Synaptic Plasticity With Online Reinforcement Learning." *Frontiers in Neurorobotics*.

10. Guillaume Bellec*, Darjan Salaj*, Anand Subramoney*, Robert Legenstein, Wolfgang Maass (2018). "Long short-term memory and Learning-to-learn in networks of spiking neurons." *Advances in Neural Information Processing Systems 31*.

11. Jacques Kaiser, Rainer Stal, Anand Subramoney, Arne Roennau, Rüdiger Dillmann (2017). "Scaling up liquid state machines to predict over address events from dynamic vision sensors." *Bioinspiration & Biomimetics.*.

12. Mihai A Petrovici, Sebastian Schmitt, Johann Klähn, David Stöckel, Anna Schroeder, Guillaume Bellec, Johannes Bill, Oliver Breitwieser, Ilja Bytschok, Andreas Grübl, Maurice Güttler, Andreas Hartel, Stephan Hartmann, Dan Husmann, Kai Husmann, Sebastian Jeltsch, Vitali Karasenko, Mitja Kleider, Christoph Koke, Alexander Kononov, Christian Mauch, Eric Müller, Paul Müller, Johannes Partzsch, Thomas Pfeil, Stefan Schiefer, Stefan Scholze, Anand Subramoney, Vasilis Thanasoulis, Bernhard Vogginger, Robert Legenstein, Wolfgang Maass, René Schüffny, Christian Mayr, Johannes Schemmel, Karlheinz Meier (2017). "Pattern representation and recognition with accelerated analog neuromorphic system." *IEEE International Symposium on Circuits and Systems (ISCAS)*.

* equal contribution

Appendix B

Appendix to Chapter 2: Spike-frequency adaptation provides a long short-term memory to networks of spiking neurons

B.1 Adaptation index

The adaptation index (AI) is a quantitative measure of firing rate adaptation that has been recorded for a wide variety of cells in the Allen institute database Allen Institute, 2018. It measures the rate at which firing of a spiking neuron speeds up or slows down when the neuron is fed with a step current of 1 second. Given the induced spike times, it is defined as:

$$\frac{1}{N-1}\sum_{n=1}^{N-1}\frac{\mathrm{ISI}_{n+1}-\mathrm{ISI}_n}{\mathrm{ISI}_{n+1}+\mathrm{ISI}_n},$$

where ISI_n is n-th inter spike interval (ISI) and N is the number of ISIs induced during the stimulus duration. Hence regular doubling of the ISI produces for example $\mathrm{AI} = 0.33$.

B.2 Network models

Leaky integrate and fire (LIF) neurons. A LIF neuron j spikes as soon at its membrane potential $V_j(t)$ is above its threshold v_{th}. At each spike time t, the membrane potential $V_j(t)$ is reset by subtracting the threshold value v_{th} and the neuron enters a strict refractory period for 2 to 5 ms (depending on the experiment) where it cannot spike again. Between spikes the membrane voltage $V_j(t)$ is following the dynamic:

$$\tau_m \dot{V}_j(t) \quad = \quad -V_j(t) + R_m I_j(t).$$

Our simulations were performed in discrete time with a time step $\delta t = 1$ ms. In discrete time, the input and output spike trains are modeled as binary sequences $x_i(t), z_j(t) \in \{0, \frac{1}{\delta t}\}$ respectively. 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.

During the refractory period following a spike, $z_j(t)$ is fixed to 0. The neural dynamics in discrete time reads as follows:

$$V_j(t+\delta t) \quad = \quad \alpha V_j(t) + (1-\alpha) R_m I_j(t) - v_{\text{th}} z_j(t) \delta t, \tag{B.1}$$

where $\alpha = \exp(-\frac{\delta t}{\tau_m})$, with τ_m being the membrane constant of the neuron j. The spike of neuron j is defined by $z_j(t) = H\left(\frac{V_j(t)-v_{\text{th}}}{v_{\text{th}}}\right)\frac{1}{\delta t}$, with $H(x) = 0$ if $x < 0$ and 1 otherwise. The term $-v_{\text{th}} z_j(t)\delta t$ implements the reset of the membrane voltage after each spike.

In all simulations the R_m was set to 1 GΩ. The input current $I_j(t)$ is defined as the weighted sum of spikes from external inputs and other neurons in the network:

$$I_j(t) \quad = \quad \sum_i W^{in}_{ji} x_i(t - d^{in}_{ji}) + \sum_i W^{rec}_{ji} z_i(t - d^{rec}_{ji}), \tag{B.2}$$

where W^{in}_{ji} and W^{rec}_{ji} denote respectively the input and the recurrent synaptic weights and d^{in}_{ji} and d^{rec}_{ji} the corresponding synaptic delays.

Adaptive leaky integrate and fire (ALIF) neurons. An ALIF neuron extends the LIF neuron with an SFA mechanism. The SFA is realized by replacing the fixed threshold v_{th} with the adaptive threshold $A_j(t)$ which follows the dynamic described in equation (2.1). The spiking output of ALIF neuron j is then defined by $z_j(t) = H(\frac{V_j(t)-A_j(t)}{A_j(t)})\frac{1}{\delta t}$.

Adaptation time constants of ALIF neurons were chosen to match the task requirements while still conforming to the experimental data from rodents Allen Institute, 2018; Pozzorini et al., 2013; Pozzorini et al., 2015; Mensi et al., 2012. For an analysis of the impact of the adaptation time constants on the performance see Table S1 in Supplement.

LIF neurons whose excitability gets increased through their firing: ELIF neurons. There exists experimental evidence that some neurons fire for the same stimulus more for a repetition of the same sensory stimulus. We refer to such neurons as ELIF neurons, since they are becoming more excitable. Such repetition enhancement was discussed for example in Tartaglia et al., 2015. But to the best of our knowledge, it has remained open whether repetition enhancement is a network effect, resulting for example from a transient depression of inhibitory synapses onto the cell that is caused by postsynaptic firing Kullmann et al., 2012, or a result of an intrinsic firing property of some neurons. We used a simple model for ELIF neurons that is dual to the ALIF neuron model: The threshold is lowered through each spike of the neuron, and then decays exponentially back to its resting value. This can be achieved by using a negative value for β in equation (2.1).

Models for Short-Term Plasticity (STP) of synapses. We modelled the STP dynamic according to the classical model of STP in Mongillo et al., 2008. The STP

dynamics in discrete time, derived from the equations in Mongillo et al., 2008, are as follows:

$$u'_{ji}(t+\delta t) = \exp\left(\frac{-\delta t}{F}\right) u'_{ji}(t) + U_{ji}(1-u_{ji}(t))z_i(t)\delta t, \tag{B.3}$$

$$u_{ji}(t+\delta t) = U_{ji} + u'_{ji}(t), \tag{B.4}$$

$$r'_{ji}(t+\delta t) = \exp\left(\frac{-\delta t}{D}\right) r'_{ji}(t) + u_{ji}(t)(1-r'_{ji}(t))z_i(t)\delta t, \tag{B.5}$$

$$r_{ji}(t+\delta t) = 1 - r'_{ji}(t), \tag{B.6}$$

$$W^{STP}_{ji}(t+\delta t) = W^{rec}_{ji} u_{ji}(t) r_{ji}(t), \tag{B.7}$$

where $z_i(t)$ is the spike train of the pre-synaptic neuron and W^{rec}_{ji} scales the synaptic efficacy of synapses from neuron i to neuron j. Networks with STP were constructed from LIF neurons with the weight W^{rec}_{ji} in equation (B.2) replaced by the time dependent weight $W^{STP}_{ji}(t)$.

STP time constants of facilitation-dominant and depression-dominant network models were based on values of experimental recordings in Y. Wang et al., 2006 of PFC-E1 and PFC-E2 synapse types respectively. Recordings in Y. Wang et al., 2006 were performed in medial prefrontal cortex of young adult ferrets. For the STORE-RECALL task, both facilitation and depression time constants were equally scaled up until the larger time constant matched the requirement of the task (see section on "Comparing networks with different slow processes" below). In the sMNIST task for the depression-dominant network model (STP-D) we used values based on PFC-E2: $F = 20$ ms, $D = 700$ ms and $U = 0.2$, and for facilitation-dominant network model (STP-F) we used values based on PFC-E1: $F = 500$ ms, $D = 200$ ms and $U = 0.2$.

Weight initialization. Initial input and recurrent weights were drawn from a Gaussian distribution $W_{ji} \sim \frac{w_0}{\sqrt{n_{in}}}\mathcal{N}(0,1)$, where n_{in} is the number of afferent neurons and $\mathcal{N}(0,1)$ is the zero-mean unit-variance Gaussian distribution and $w_0 = \frac{1\,\text{Volt}}{R_m}\delta t$ is a normalization constant Bellec et al., 2018b.

B.3 Training methods

BPTT. In artificial recurrent neural networks such as LSTMs, gradients can be computed with backpropagation through time (*BPTT*). In spiking neural networks, complications arise from the non-differentiability of the output of spiking neurons. In our discrete time simulation, this is formalized by the discontinuous step function H arising in the definition of the spike variable $z_j(t)$. All other operations can be differentiated exactly with *BPTT*. For feedforward artificial neural networks using step functions, a solution was to use a pseudo derivative $H'(x) := \max\{0, 1 - |x|\}$, but we observed that this convention is unstable with recurrently connected

neurons. We found that dampening this pseudo-derivative with a factor $\gamma < 1$ (typically $\gamma = 0.3$) solves that issue. Hence we use the pseudo-derivative:

$$\frac{dz_j(t)}{dv_j(t)} := \gamma \max\{0, 1 - |v_j(t)|\}, \tag{B.8}$$

where $v_j(t)$ denotes the normalized membrane potential $v_j(t) = \frac{V_j(t) - A_j(t)}{A_j(t)}$. Importantly, gradients can propagate in adaptive neurons through many time steps in the dynamic threshold without being affected by the dampening.

e-prop. In the 12AX task the networks were trained using biologically plausible learning method *random e-prop* Bellec et al., 2019b in addition to *BPTT*.

B.4 Tasks

The STORE-RECALL task of Fig. 2.2 The input to the network consisted of STORE, RECALL, and 20 bits which were represented by sub-populations of spiking input neurons. STORE and RECALL commands were represented by 4 neurons each. The 20 bits were represented by population coding where each bit was assigned 4 input neurons (2 for value zero, and 2 for value one). When a sub-population is active, it would exhibit a Poisson firing with frequency of 400 Hz. To measure the generalization capability of a trained network, we first generate a test set dictionary of 20 unique feature vectors (random bit strings of length 20) that have at least a Hamming distance of 5 bits among each other. For every training batch a new dictionary of 40 random bit strings (of length 20) would be generated where each string has a Hamming distance of at least 5 bits from any of the bit string in the test set dictionary. This way we ensure that, during training, a network never encounters any bit string similar to one from the test set. Each input sequence consisted of 10 steps (200 ms each) where a different population encoded bit string is shown during every step. Only during the RECALL period, the 20 bit input populations are silent. At every step, the STORE or the RECALL populations were activated interchangeably with probability 0.2 which resulted in distribution of delays between the STORE-RECALL pairs in the range [200, 1600] ms.

The training and the test performance were computed as average over 256 and 512 random input sequences respectively. Networks were trained for 4000 iterations and stopped if the error on the training batch was below 1%. We used the Adam optimizer with default parameters and initial learning rate of 0.01 which is decayed every 200 iterations by a factor of 0.8. We also used learning rate ramping, which, for the first 200 iterations, monotonically increased the learning rate from 0.00001 to 0.01. To avoid unrealistically high firing rates, the loss function contained a regularization term (scaled with coefficient 0.001) that minimizes the squared difference of the average firing rate of individual neurons from a target firing rate of 10 Hz. To improve convergence, we also included an entropy component to the

loss (scaled with coefficient 0.3) which was computed as the mean of the entropies of the sigmoid neurons outputs.

We trained an ALIF network and a LIF network, both consisting of 500 recurrently connected neurons. The membrane time constant was $\tau_m = 20\,\text{ms}$. For adaptation parameters we used $\beta_{ALIF} = 4\,\text{mV}$ and $\tau_a = 800\,\text{ms}$ with baseline threshold voltage 10 mV. Synaptic delay was 1 ms. The input to the sigmoidal readout neurons were the neuron traces that were calculated by passing all the network spikes through a low-pass filter with a time constant of 20 ms.

We ran 5 training runs with different random seeds for both ALIF and LIF network models. All runs of the ALIF network converged after $\sim$ 3600 iterations to a training error below 1%. At that point we measured the accuracy on 512 test sequences generated using the previously unseen test bit strings which resulted in test accuracy of 99.09% with standard deviation of 0.17%. The LIF network was not able to solve the task in any of the runs (all runs resulted in 0% training and test accuracy with zero standard deviation). On the level of individual feature recall accuracy, the best out of 5 training runs of the LIF network was able to achieve 49% accuracy which is the chance level since individual features are binary bits. In contrast, all ALIF network runs had individual feature level accuracy of above 99.99%.

Decoding memory from the network activity. We trained a Support Vector Machine (SVM) to classify the stored memory content from the network spiking activity in the step before the RECALL (200 ms before the start of RECALL command). We performed a cross-validated grid-search to find the best hyperparameters for the SVM which included kernel type [linear, polynomial, RBF] and penalty parameter C of the error term [0.1, 1, 10, 100, 1000]. We trained SVMs on test batches of the 5 different training runs (see above). SVMs trained on the period preceding the RECALL command of a test batch achieved an average of 4.38% accuracy with standard deviation of 1.29%. In contrast SVMs trained on a period during the RECALL command achieved an accuracy of 100%. This demonstrates that the memory stored in the network is not decodable from the network firing activity before the RECALL input command.

Additionally, analogous to the experiments of Wolff et al., 2017, we trained SVMs on network activity during the encoding (STORE) period and evaluated them on the network activity during reactivation (RECALL), and vice versa. In both scenarios, the classifiers were not able to classify the memory content of the evaluation period (0.0% accuracy).

Comparing networks with different slow processes on a simplified version of the STORE-RECALL task. For the comprehensive comparison of networks endowed with different slow processes in neuron and synapse dynamics we used a single dimensional version of the STORE-RECALL task where only a single feature needs to be stored and recalled from memory. The input to the network consisted of

40 input neurons: 10 for STORE, 10 for RECALL, and 20 for population coding of a binary feature. Each sequence consisted of 20 steps (200 ms each) where the STORE or the RECALL populations were activated with probability 0.09 interchangeably which resulted in delays between the STORE-RECALL pairs to be in the range [200, 3600] ms.

The training batch and the test performance were computed as average over 128 and 2048 random input sequences respectively. All networks were trained for 400 iterations. We used the Adam optimizer with default parameters and initial learning rate of 0.01 which was decayed every 100 iterations by a factor of 0.3. The same firing rate regularization term was added to the loss as in the original STORE-RECALL setup (see above).

All networks consisted of 60 recurrently connected neurons. The membrane time constant was $\tau_m = 20\,\text{ms}$. For ALIF and ELIF networks, we used $\beta_{ALIF} = 1\,\text{mV}$ and $\beta_{ELIF} = -0.5\,\text{mV}$ with $\tau_a = 2000\,\text{ms}$. Synapse parameters of STP-D network were $F = 51 \pm 15$ ms, $D = 2000 \pm 51$ ms and $U = 0.25$, and of STP-F network $F = 2000 \pm 146$ ms, $D = 765 \pm 71$ ms and $U = 0.28$. The baseline threshold voltage was 10 mV for all models except ELIF for which it was 20 mV. Synaptic delay was 1 ms across all network models.

Google Speech Commands task. Features were extracted from the raw audio using the Mel Frequency Cepstral Coefficient (MFCC) method with 30 ms window size, 1 ms stride and 40 output features. The network models were trained to classify the input features to one of the 10 keywords (yes, no, up, down, left, right, on, off, stop, go) or to two special classes for silence or unknown word (where the remainder of 20 recorded keywords are grouped). The training, validation and test set were assigned 80, 10, and 10 percent of data respectively while making sure that audio clips from the same person stay in the same set.

All networks were trained for 18,000 iterations using the Adam optimizer with batch size 100. The output spikes of the networks were averaged over time, and the linear readout layer was applied to those values. During the first 15,000 we used a learning rate of 0.001 and for the last 3000 we used a learning rate of 0.0001. The loss function contained a regularization term (scaled with coefficient 0.001) that minimizes the squared difference of average firing rate between individual neurons and a target firing rate of 10 Hz.

Both ALIF and LIF networks consisted of 2048 fully connected neurons in a single recurrent layer. The neurons had a membrane time constant of $\tau_m = 20\,\text{ms}$, the adaptation time constant of ALIF neurons was $\tau_a = 100\,\text{ms}$, adaptation strength was $\beta = 2\,\text{mV}$. The baseline threshold was $v_{\text{th}} = 10\,\text{mV}$, and the refractory period was 2 ms. Synaptic delay was 1 ms.

Delayed-memory XOR task. The pulses on the two input channels were generated with 30 ms duration and the shape of a normal probability density function

normalized in the range $[0, 1]$. The pulses were added or subtracted from the baseline zero input current at appropriate delays. The go-cue was always a positive current pulse. The 6 possible configurations of the input pulses ($+$, $-$, $++$, $--$, $+-$, $-+$) were sampled with equal probability during training and testing.

Networks were trained for 2000 iterations using the Adam optimizer with batch size 256. The initial learning rate was 0.01 and every 200 iterations the learning rate was decayed by a factor of 0.8. The loss function contained a regularization term (scaled with coefficient 50) that minimizes the squared difference of the average firing rate of individual neurons from a target firing rate of 10 Hz. This regularization resulted in networks with mean firing rate of 10 Hz where firing rates of individual neurons were spread in the range [1, 16] Hz.

Both ALIF and LIF networks consisted of 80 fully connected neurons in a single recurrent layer. The neurons had a membrane time constant of $\tau_m = 20\,\text{ms}$, a baseline threshold $v_{\text{th}} = 10\,\text{mV}$, and a refractory period of $3\,\text{ms}$. ALIF neurons had an adaptation time constant of $\tau_a = 500\,\text{ms}$ and an adaptation strength of $\beta = 1\,\text{mV}$. Synaptic delay was $1\,\text{ms}$. For training the network to classify the input into one of the three classes, we used the cross-entropy loss between the labels and the softmax of three linear readout neurons. The input to the linear readout neurons were the neuron traces that were calculated by passing all the network spikes through a low-pass filter with a time constant of $20\,\text{ms}$.

The sequential MNIST (sMNIST) task. The input consisted of sequences of 784 pixel values created by unrolling the handwritten digits of the MNIST dataset, one pixel after the other in a scanline manner as indicated in Fig. S3A. For comparing different spiking network models, we used 1 ms presentation time for each pixel (Fig. 2.3C). LSTM networks also work well for tasks on larger time-scales. Hence for comparing LSNNs with LSTM networks, we used a version of the task with 2 ms presentation time per pixel, thereby doubling the length of sequences to be classified to 1568 ms (Fig. 2.3D). A trial of a trained LSNN (with an input sequence that encodes a handwritten digit "3" using population rate coding) is shown in Fig. S3B. The top row of Fig. S3B shows a version where the grey value of the currently presented pixel is encoded by population coding, through the firing probability of 80 input neurons. Somewhat better performance was achieved when each of the 80 input neurons was associated with a particular threshold for the grey value, and this input neuron fired whenever the grey value crossed its threshold in the transition from the previous to the current pixel (this input convention was used for the results of Fig. 2.3C,D). Grey values of pixels were presented to the LSTM network simply as analog numbers.

Networks were trained for 36,000 iterations using the Adam optimizer with batch size 256. The initial learning rate was 0.01 and every 2500 iterations the learning rate was decayed by a factor of 0.8. The loss function contained a regularization term (scaled with coefficient 0.1) that minimizes the squared difference of average firing rate between individual neurons and a target firing rate of 10 Hz.

The neurons had a membrane time constant of $\tau_m = 20\,\text{ms}$, a baseline threshold of $v_{\text{th}} = 10\,\text{mV}$, and a refractory period of 5 ms. The adaptation time constants of ALIF and ELIF neurons were $\tau_a = 700\,\text{ms}$ in Fig. 2.3C. ALIF neurons had $\tau_a = 1400\,\text{ms}$ in Fig. 2.3D. The adaptation strength of ALIF neurons was $\beta = 1.8\,\text{mV}$, and of ELIF neurons $\beta = -0.9\,\text{mV}$. Synaptic delay was 1 ms. The output of the LSNN is produced by the softmax of 10 linear output neurons that receive spikes from all neurons in the network, as shown in the bottom row of Fig. S3B. For training the network to classify to one of the ten classes we used cross-entropy loss computed between the labels and the softmax of output neurons. The input to the linear readout neurons were the neuron traces that were calculated by passing all the network spikes through a low-pass filter with a time constant of 20 ms.

The 12AX task. The input for each training and testing episode consisted of a sequence of 90 symbols from the set {1,2,A,B,C,X,Y,Z}. A single episode could contain multiple occurrences of digits 1 or 2 (up to 23), each time changing the target sequence (A...X or B...Y) after which the network was supposed to output R. Each digit could be followed by up to 26 letters before the next digit appeared. More precisely, the following regular expression describes the string that is produced: `[12][ABCXYZ]{1,10}((A[CZ]{0,6}X|B[CZ]{0,6}Y)|([ABC][XYZ])){1,2}`. Each choice in this regular expression is made randomly.

The neurons had a membrane time constant of $\tau_m = 20\,\text{ms}$, a baseline threshold $v_{\text{th}} = 30\,\text{mV}$, a refractory period of 5 ms, and synaptic delays of 1 ms. ALIF neurons had an adaptation strength of $\beta = 1.7\,\text{mV}$, and adaptation time constants were chosen uniformly from $[1, 13500]$ ms.

A cross-entropy loss function was used along with a regularization term (scaled with coefficient 15) that minimizes the squared difference of average firing rate between individual neurons and a target firing rate of 10 Hz. The LSNN was trained for 10,000 iterations with a batch size of 20 episodes and a fixed learning rate of 0.001. An episode consisted of 90 steps, with between 4 to 23 tasks generated according to the task generation procedure described previously. We trained the network with *BPTT* using 5 different seeds, which resulted in average test success rate 97.79% with standard deviation 0.42%. The network trained with *random e-prop* using 5 different seeds resulted in average test success rate 92.89% with standard deviation 0.75%.

Symbolic computation on strings of symbols. The input to the network consisted of 35 symbols - 31 symbols represented symbols from the English alphabet {a, b, c, d, ... x, y, z, A, B, C, D, E}, one symbol was for "end-of-string" (EOS) '*', one for cue for the output prompt '?', and two symbols to denote whether the task instruction was duplication or reversal. The task and the rest of the symbols were encoded using separate one-hot vectors of dimension 2 and 33 respectively. Inputs to the network were transformed into spike trains using a population of 5 spiking neurons for each input component for a total of 175 input neurons. This population

fired at a "high" rate (200 Hz) to encode 1, and at a "low" rate (2 Hz) otherwise. The output consisted of 32 linear readouts: 31 for symbols from the English alphabet and one additional readout for the "end-of-string" symbol. The input to these linear readouts was the value of neuron traces at the end of each step of 500 ms during the output period, i.e, the second half of each episode. The neuron traces were calculated by passing all the network spikes through a low-pass filter with a time constant of 250 ms. The final output symbol was produced using the argmax over the value of all the readouts (a softmax instead of the hard argmax was used during training). The network was trained to minimize the cross entropy error between the softmax applied on the output layer and targets. The loss function contained a regularization term (scaled with coefficient 5) that minimizes the squared difference of average firing rate between individual neurons and a target firing rate of 20 Hz.

The training was performed for 50,000 iterations, with a batch size of 50 episodes. We used the Adam optimizer with default parameters and fixed learning rate of 0.001. Each symbol was presented to the network for a duration of 500 ms. The primary metric we used for measuring the performance of the network was success rate, which was defined as the percentage of episodes where the network produced the full correct output for a given string i.e. all the output symbols in the episode had to be correct. The network was tested on 50,000 previously unseen strings.

The network consisted of 128 LIF and 192 ALIF neurons. All the neurons had a membrane time constant of $\tau_m = 20\,\text{ms}$, a baseline threshold $v_{\text{th}} = 30\,\text{mV}$, a refractory period of 5 ms, and a synaptic delay of 1 ms. ALIF neurons in the network had an adaptation strength of $\beta = 1.7\,\text{mV}$, and an adaptation time constant randomly uniformly chosen from the range $[1, 6000]$ ms. All other parameters were the same as in the other experiments. We trained the network using 5 different seeds and tested it on previously unseen strings. Average test success rate was 95.88% with standard deviation 1.39%.

Analysis of spiking data. We used 3-way ANOVA to analyze if a neuron's firing rate is significantly affected by task, serial position in the sequence, symbol identity, or combination of these (similar to Lindsay et al., 2017). We refer to these factors as "conditions". The analysis was performed on the activity of the neurons of the trained LSNN during 50,000 test episodes. For the analysis, neurons whose average firing rate over all episodes was lower than 2Hz or greater than 60Hz were discarded from the analysis to remove large outliers. This left 279 out of the 320 neurons. From each episode, a serial position from the input period was chosen randomly, and hence each episode could be used only once, i.e., as one data point. This was to make sure that each entry in the 3-way ANOVA was completely independent of other entries, since the neuron activity within an episode is highly correlated. Each data point was labeled with the corresponding triple of (task type, serial position, symbol identity). To ensure that the dataset was balanced, the same number of data points per particular combination of conditions was used, discarding all the excess data points, resulting in a total of 41,850 data points.

To categorize a neuron as selective to one or more conditions, or combination of conditions, we observed p-values obtained from 3−way ANOVA and calculated the effect size ω^2 for each combination of conditions. If the p-value was smaller than 0.001 and ω^2 greater than 0.14 for a particular combination of conditions, the neuron was categorized as selective to that combination of conditions. The ω^2 threshold of 0.14 was suggested by Field, 2013 to select large effect sizes. Each neuron can have large effect size for more than one combination of conditions. Thus the values shown in Fig. 2.5D sum to > 1. The neuron shown in Fig. 2.5E had the most prominent selectivity for the combination of Task $\times$ Position $\times$ Symbol, with $\omega^2 = 0.394$ and $p < 0.001$. The neuron shown in Fig. 2.5F was categorized as selective to a combination of Position $\times$ Symbol category, with $\omega^2 = 0.467$ and $p < 0.001$. While the 3-way ANOVA tells us if a neuron is selective to a particular combination of conditions, it does not give us the exact task/symbol/position that the neuron is selective to. To find the specific task/symbol/position that the neuron was selective to, Welch's t-test was performed, and a particular combination with maximum t-statistic and $p < 0.001$ was chosen to be shown in Fig. 2.5E,F.

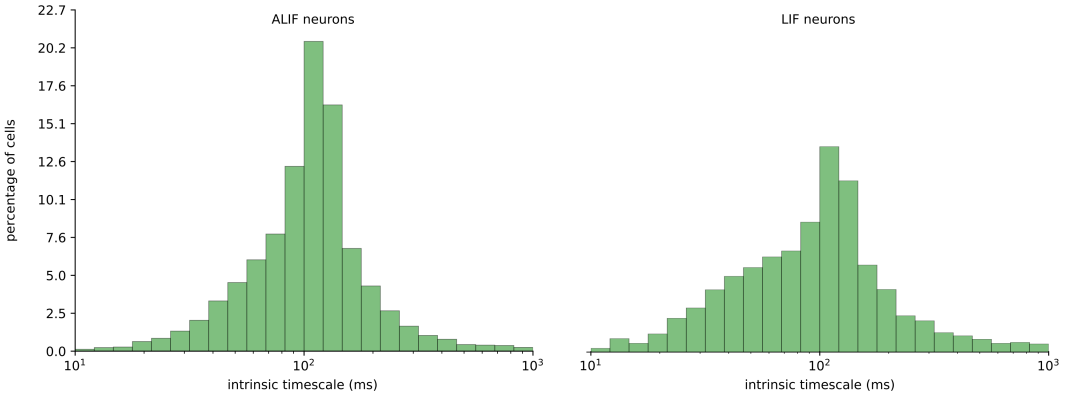

Figure B.1: Histogram of intrinsic timescale of LSNN. We trained 64 randomly initialized LSNNs consisting of 200 LIF and 200 ALIF neurons on the single-feature STORE-RECALL task. Measurements of intrinsic timescale were performed according to Wasmuht et al., 2018 on the spiking data of LSNNs solving the task after training. Averaged data of all 64 runs is presented in histogram. The distribution is very similar for LIF and ALIF neurons.

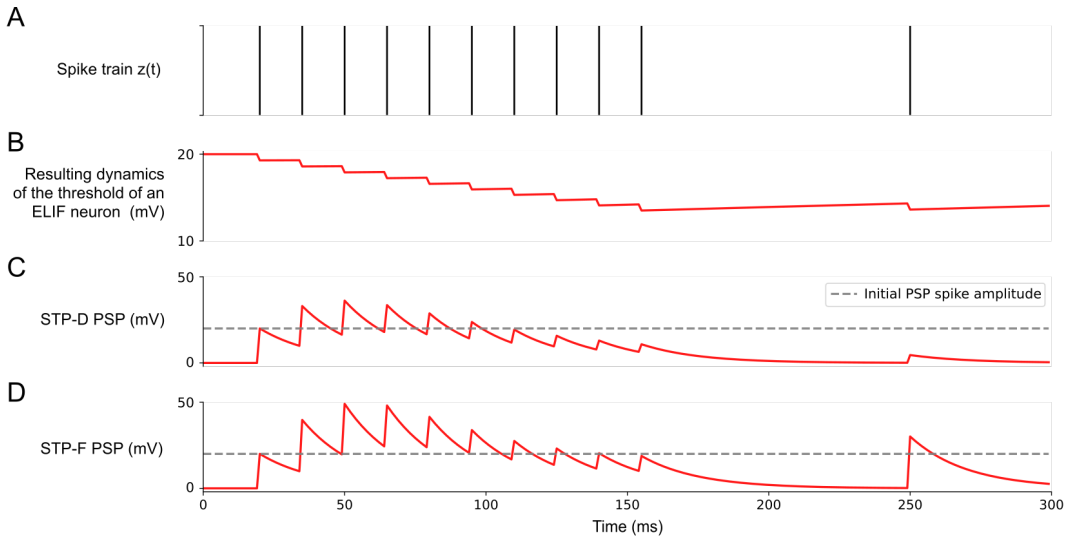

Figure B.2: Illustration of models for an inversely adapting ELIF neuron, and for short-term synaptic plasticity. (**A**) Sample spike train. (**B**) Resulting evolution of firing threshold for an inversely adapting neuron (ELIF neuron). (**C**-**D**) Resulting evolution of the amplitude of post synaptic potentials (PSPs) for spikes of the presynaptic neuron for the case of a depression-dominant (STP-D: D$\gg$F) and a facilitation-dominant (STP-F: F$\gg$D) short-term synaptic plasticity.

Expected delay between STORE and RECALL	200ms	2s	4s	8s	16s
$\tau_a = 200$ms	0.8	26.4	42	49	49
$\tau_a = 2$s	1	0.4	1.2	7.8	24.8
$\tau_a = 4$s	0.9	0.3	0.3	2.2	9.5
$\tau_a = 8$s	0.4	0.2	0.3	2.3	2.9
τ_a power dist. in [0, 8]s	0.4	0.3	1.6	3.7	16.4
τ_a uniform dist. in [0, 8]s	3.8	0.1	1.4	7.9	7.4

Table B.1: Performance (% of errors) of LSNNs with different time constants of ALIF neurons for variations of the STORE-RECALL task with different expected delays. One sees that good task performance does not require good alignment of adaptation time constants of ALIF neurons with required duration of working memory. An LSNN network of 60 ALIF neurons was trained in all different configurations of adaptation time constant and expected delay of a single dimensional STORE-RECALL task. Every step in the input sequence had a duration of 200 ms. An exception is the task instance with expected delay of 200 ms where each step had a duration of 50 ms. Training was performed with a batch size of 64. All other parameters match the description of single dimensional STORE-RECALL task in Results section "Working memory performance of variants of SNNs with other slow processes in neurons or synapses". A network of 60 LIF neurons trained under the same parameters did not improve beyond chance level ($\sim$ 50% error), except on the task instance with expected delay of 200 ms where the LIF network reached 3.3% error.

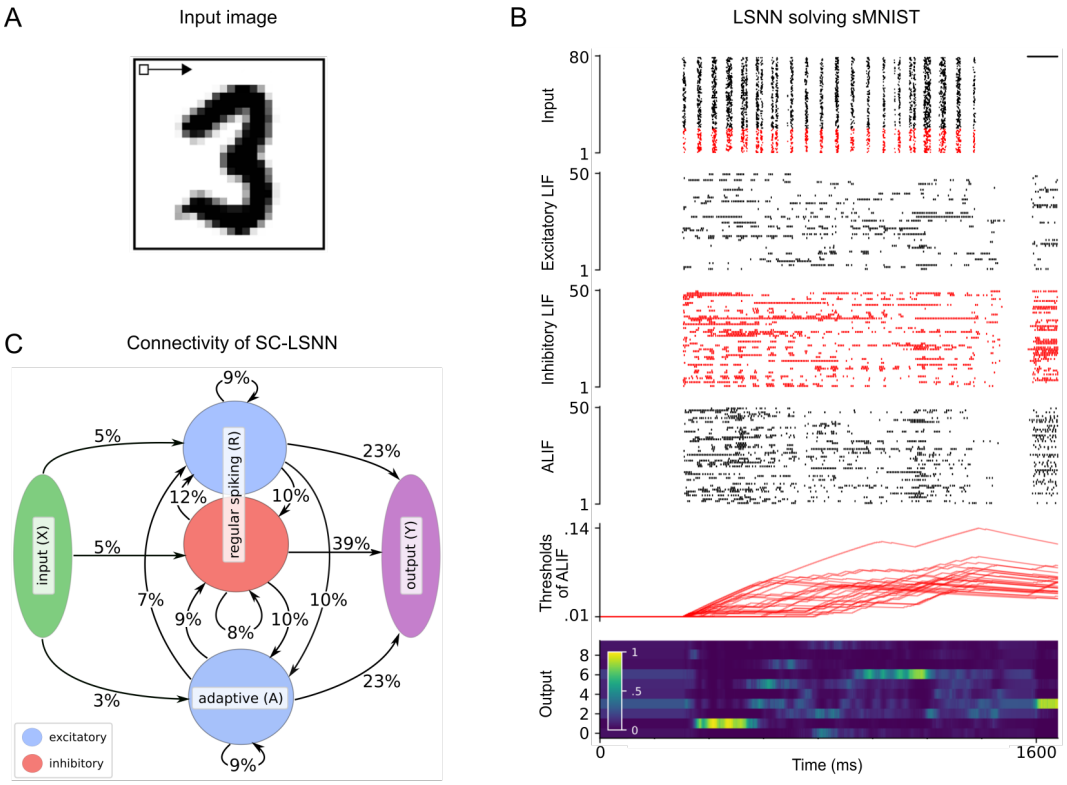

Figure B.3: sMNIST time series classification benchmark task. All performance data are reported for test samples that did not occur during training. (**A**) Illustration of the pixel-wise input presentation of handwritten digits for sMNIST. (**B**) Rows top to bottom: Input encoding for an instance of the sMNIST task, network activity, and temporal evolution of firing thresholds for randomly chosen subsets of LIF and ALIF neurons in the SC-LSNN version, where 25% of the LIF neurons were inhibitory (their spikes are marked in red). The light color of the readout neuron for digit "3" around 1600 ms indicates that this input was correctly classified. (**C**) Resulting connectivity graph between neuron populations of an SC-LSNN after BPTT optimization with DEEP R on sMNIST task with 12% global connectivity limit.

Appendix C

Appendix to Chapter 3: Reservoirs learn to learn

C.1 Leaky integrate and fire neurons

We used leaky integrate-and-fire (LIF) models of spiking neurons, where the membrane potential $V_j(t)$ of neuron j evolves according to:

$$V_j(t+1) = \rho_j V_j(t) + (1-\rho_j) R_m I_j(t) - B_j(t) z_j(t) \tag{C.1}$$

where R_m is the membrane resistance, and ρ_j is the decay constant defined using the membrane time constant τ_j as $\rho_j = e^{\frac{-\Delta t}{\tau_j}}$, where Δt is the time step of simulation. A neuron j spikes as soon at its normalized membrane potential $v_j(t) = \frac{V_j(t)-B_j(t)}{B_j(t)}$ is above its firing threshold v_{th}. At each spike time t, the membrane potential $V_j(t)$ is reset by subtracting the current threshold value $B_j(t)$. After each spike, the neuron enters a strict refractory period during which it cannot spike.

C.2 Backpropagation through time

We introduced a version of backpropagation through time (BPTT) in Bellec et al., 2018b which allows us to back-propagate the gradient through the discontinuous firing event of spiking neurons. The firing is formalized through a binary step function H applied to the scaled membrane voltage $v(t)$. The gradient is propagated through this step function with a pseudo-derivative as in Courbariaux et al., 2016; Esser et al., 2016, but with a dampened amplitude at each spike.

Specifically, the derivative of the spiking $z_j(t)$ with respect to the normalized membrane potential $v_j(t) = \frac{V_j(t)-B_j(t)}{B_j(t)}$ is defined as:

$$\frac{dz_j(t)}{dv_j(t)} := \gamma \max\{0, 1-|v_j(t)|\}. \tag{C.2}$$

In this way the architecture and parameters of an RSNN can be optimized for a given computational task.

C.3 Optimizing reservoirs to learn

Reservoir model: Our reservoir consisted of 800 recurrently connected leaky integrate-and-fire (LIF) neurons according to the dynamics defined above. The network simulation is carried out in discrete timesteps of $\Delta t = 1\,\text{ms}$. The membrane voltage decay was uniform across all neurons and was computed to correspond to a time constant of 20 ms ($\rho_j = 0.368$). The normalized spike threshold was set to 0.02 and a refractory period of 5 ms was introduced. Synapses had delays of 5 ms. In the beginning of the experiment, input W^{in} and recurrent weights W^{rec} were initialized according to Gaussian distributions with zero mean and standard deviations of $\frac{1}{\sqrt{3}}$ and $\frac{1}{\sqrt{800}}$ respectively. Similarly, the initial values of the readout $W^{\text{out,init}}$ were also optimized in the outer loop, and were randomly initialized in the beginning of the experiment according to a uniform distribution, as proposed in Glorot and Bengio, 2010.

Readout learning: The readout was iteratively adapted according to equation (3.2). It received as input the input $\boldsymbol{x}_C(t)$ itself and the features $\boldsymbol{h}_C(t)$ from the reservoir, which were given as exponentially filtered spike trains: $h_{C,j}(t) = \sum_{t' \leq t} \kappa^{t-t'} z_{C,j}(t')$. Here, $\kappa = e^{\frac{-\Delta t}{\tau_{\text{readout}}}}$ is the decay of leaky readout neurons. Weight changes were computed at each timestep and accumulated. After every second these changes were used to actually modify the readout weights. Thus, formulated in discrete time, the plasticity of the readout weights in a task C took the following form:

$$\Delta W_C^{\text{out}} = \eta \sum_{t'=t-1000\,\text{ms}}^{t} \left(\boldsymbol{y}_C(t') - \widehat{\boldsymbol{y}}_C(t') \right) \cdot \boldsymbol{h}_C(t')^T, \tag{C.3}$$

where η is a learning rate.

Outer loop optimization: To optimize input and recurrent weights of the reservoir in the outer loop, we simulated the learning procedure described above for $m = 40$ different tasks in parallel. After each 3 seconds, the simulation was paused and the outer loop objective was evaluated. Note that the readout weights were updated 3 times within these 3 seconds according to our scheme. The outer loop objective, as given in equation 3.3, is approximated by:

$$\mathcal{L} = \frac{1}{m} \sum_{n=1}^{m} \sum_{t'=t-2000\,\text{ms}}^{t} \left\| \boldsymbol{y}_n(t') - \widehat{\boldsymbol{y}}_n(t') \right\|_2^2 + \mathcal{L}_{\text{reg}} \,. \tag{C.4}$$

We found that learning is improved if one includes only the last two seconds of simulation. This is because the readout weights seem fixed and unaffected by the plasticity of equation 3.2 in the first second, as BPTT cannot see beyond the truncation of 3 seconds. The cost function $\mathcal{L}$ was then minimized using a variant of gradient descent (Adam Kingma and Ba, 2014), where a learning rate of 0.001

was used. The required gradient $\nabla\mathcal{L}$ was computed with BPTT using the 3 second chunks of simulation and was clipped if the $\mathcal{L}_2$-norm exceeded a value of 1000.

Regularization: In order to encourage the model to settle into a regime of plausible firing rates, we add to the outer loop cost function a term that penalizes excessive firing rates:

$$\mathcal{L}_{\text{reg}} = \alpha \sum_{j=1}^{800} (f_j - 20\,\text{Hz})^2, \tag{C.5}$$

with the hyperparameter $\alpha = 1200$. We compute the firing rate of a neuron f_j based on the number of spikes in the past 3 seconds.

Task details: We describe here the procedure according to which the input time series $\boldsymbol{x}_C(t)$ and target time series $\boldsymbol{y}_C(t)$ were generated. The input signal was composed of a sum of two sines with random phase $\phi_n \in [0, \frac{\pi}{2}]$ and amplitude $A_n \in [0.5, 1]$, both sampled uniformly in the given interval.

$$\boldsymbol{x}_C(t) = \sum_{n=1}^{2} A_n \sin(2\pi \frac{t}{T_n} + \phi_n), \tag{C.6}$$

with periods of $T_1 = 0.323\,\text{s}$ and $T_2 = 0.5\,\text{s}$.

The corresponding target function $\boldsymbol{y}_C(t)$ was then computed by an application of a random second order Volterra filter to $\boldsymbol{x}_C(t)$ according to equation (3.5). Each task uses a different kernel in the Volterra filter and we explain here the process by which we generate the kernels k^1 and k^2. Recall that we truncate the kernels after a time lag of 500 ms. Together with the fact that we simulate in discrete time steps of 1 ms we can represent k^1 as a vector with 500 entries, and k^2 as a matrix of dimension 500×500.

Sampling k^1: We parametrize k^1 as a normalized sum of two different exponential filters with random properties:

$$\tilde{k}^1(t) = \sum_{n=1}^{2} a_n \exp\left(-\frac{t}{b_n}\right), \tag{C.7}$$

$$k^1(t) = \frac{\tilde{k}^1(t)}{\|\tilde{k}^1\|_1}, \tag{C.8}$$

with a_n being sampled uniformly in $[-1, 1]$, and b_n drawn randomly in $[0.1\,\text{s}, 0.3\,\text{s}]$. For normalization, we use the sum of all entries of the filter in the discrete representation ($t \in \{0, 0.001, 0.002, \ldots, 0.499\}$).

Sampling k^2: We construct k^2 to resemble a Gaussian bell shape centered at $t = 0$, with a randomized "covariance" matrix Σ, which we parametrize such that we always obtain a positive definite matrix:

$$\Sigma = \begin{bmatrix} \sqrt{1+u^2+v^2}+u & v \\ v & \sqrt{1+u^2+v^2}-u \end{bmatrix}, \tag{C.9}$$

where u, v are sampled uniformly in $[-12, 12]$. With this we defined the kernel k^2 according to:

$$\tilde{k}^2(t_1, t_2) = \exp\left(-\frac{1}{24}\left[t_1, t_2\right]\Sigma^{-1}\begin{bmatrix} t_1 \\ t_2 \end{bmatrix}\right), \tag{C.10}$$

$$k^2(t_1, t_2) = \frac{\tilde{k}^2(t_1, t_2)}{\|\tilde{k}^2\|_1} \cdot 14. \tag{C.11}$$

The normalization term here is again given by the sum of all entries of the matrix in the discrete time representation ($[t_1, t_2] \in \{0, 0.001, 0.002, \ldots, 0.499\}^2$).

C.4 Reservoirs can also learn without changing synaptic weights to readout neurons

Reservoir model: The reservoir model used here was the same as that in Section C.3, but with 300 neurons.

Input encoding: Analog values were transformed into spiking trains to serve as inputs to the reservoir as follows: For each input component, 100 input neurons are assigned values $m_1, \ldots m_{100}$ evenly distributed between the minimum and maximum possible value of the input. Each input neuron has a Gaussian response field with a particular mean and standard deviation, where the means are uniformly distributed between the minimum and maximum values to be encoded, and with a constant standard deviation. More precisely, the firing rate r_i (in Hz) of each input neuron i is given by $r_i = r_{\max} \exp\left(-\frac{(m_i - z_i)^2}{2\sigma^2}\right)$, where $r_{\max} = 200$ Hz, m_i is the value assigned to that neuron, z_i is the analog value to be encoded, and $\sigma = \frac{(m_{\max} - m_{\min})}{1000}$, $m_{\min}$ with $m_{\max}$ being the minimum and maximum values to be encoded.

Setup and training schedule: The output of the reservoir was a linear readout that received as input the mean firing rate of each of the neurons per step i.e the number of spikes divided by 20 for the 20 ms time window that constitutes a step.

The network training proceeded as follows: A new target function was randomly chosen for each *episode* of training, i.e., the parameters of the target function are chosen uniformly randomly from within the ranges above.

Each *episode* consisted of a sequence of 400 *steps*, each lasting for 20 ms. In each step, one training example from the current function to be learned was presented to the reservoir. In such a step, the inputs to the reservoir consisted of a randomly chosen vector $\mathbf{x} = (x_1, x_2)$ as described earlier. In addition, at each step, the reservoir also got the target value $C(x_1', x_2')$ from the previous step, i.e., the value of the target calculated using the target function for the inputs given at the previous step (in the first step, $C(x_1', x_2')$ is set to 0). The previous target input was provided to the reservoir during all steps of the episode.

All the weights of the reservoir were updated using our variant of BPTT, once per *iteration*, where an *iteration* consisted of a batch of 10 *episodes*, and the weight updates were accumulated across episodes in an iteration. The ADAM Kingma and Ba, 2014 variant of gradient descent was used with standard parameters and a learning rate of 0.001. The loss function for training was the mean squared error (MSE) of the predictions over an iteration (i.e. over all the steps in an episode, and over the entire batch of episodes in an iteration), with the optimization problem written as:

$$\min_{\Theta} \mathbb{E}_{C\sim\mathcal{F}}\left[\sum_{t}\left(C(x_1^t, x_2^t; \Theta) - \widehat{C}(x_1^t, x_2^t; \Theta)\right)^2\right]. \tag{C.12}$$

In addition, a regularization term was used to maintain a firing rate of 20 Hz as in equation C.5, with $\alpha = 30$. In this way, we induce the reservoir to use sparse firing. We trained the reservoir for 5000 iterations.

Parameter values: The parameters of the leaky integrate-and-fire neurons were as follows: 5 ms neuronal refractory period, delays spread uniformly between $0 - 5$ ms, membrane time constant $\tau_j = \tau = 20$ms ($\rho_j = \rho = 0.368$) for all neurons j, $v_{\text{th}} = 0.03$ V baseline threshold voltage. The dampening factor for training was $\gamma = 0.4$ in equation C.2.

Comparison with Linear baseline: The linear baseline was calculated using linear regression with L2 regularization with a regularization factor of 100 (determined using grid search), using the mean spiking trace of all the neurons. The mean spiking trace was calculated as follows: First the neuron traces were calculated using an exponential kernel with 20 ms width and a time constant of 20 ms. Then, for every step, the mean value of this trace was calculated to obtain the mean spiking trace. In Fig. 3.4B, for each episode consisting of 400 steps, the mean spiking trace from a subset of 320 steps was used to train the linear regressor, and the mean spiking trace from remaining 80 steps was used to calculate the test error. The reported baseline is the mean of the test error over one batch of 1000 episodes with error bars of one standard deviation.

The total test MSE was 0.0056 ± 0.0039 (linear baseline MSE was 0.0217 ± 0.0046) for the TN task.

Comparison with random reservoir In Fig. 3.4B,C, a reservoir with randomly initialized input, recurrent and readout weights was tested in the same way as the optimized reservoir – with the same sets of inputs, and without any synaptic plasticity in the inner loop. The plotted curves are the average over 8000 different TNs.

Comparison with backprop: The comparison was done for the case where the reservoir was trained on the function family defined by target networks. A feed-forward (FF) network with 10 hidden neurons and 1 output was constructed. The input to this FF network were the analog values that were used to generate the spiking input and targets for the reservoir. Therefore the FF had 2 inputs, one for each of x_1 and x_2. The error reported in Fig. 3.4G is the mean training error over 1000 TNs with error bars of one standard deviation.

The FF network was initialized with Xavier normal initialization Glorot and Bengio, 2010 (which had the best performance, compared to Xavier uniform and plain uniform between $[-1, 1]$). Adam Kingma and Ba, 2014 with AMSGrad Reddi et al., 2018 was used with parameters $\eta = 10^{-1}, \beta_1 = 0.7, \beta_2 = 0.9, C = 10^{-5}$. These were the optimal parameters as determined by a grid search. Together with the Xavier normal initialization and the weight regularization parameter C, the training of the FF favoured small weights and biases.

Appendix D

Appendix to Chapter 4: Prior knowledge and network dynamics enable networks of spiking neurons to learn new tasks from just a few trials

D.1 Network models

Neurons are modelled after the standard adaptive leaky integrate-and-fire model as in Bellec et al., 2018b. A neuron j consists of two state variables — its membrane potential $V_j(t)$ and threshold $A_j(t)$. Whenever the membrane potential $V_j(t)$ exceeds the threshold $A_j(t)$, the neuron emits a spike z_j, and its membrane potential is reset by subtracting the threshold value v_{th} from it. After this, the neuron enters a refractory period of some τ_{ref} time steps during which time it cannot spike. Between spikes, the membrane potential $V_j(t)$ evolves according to:

$$\tau_m \dot{V}_j(t) = -V_j(t) + R_m I_j(t), \tag{D.1}$$

where R_m is the membrane resistance, and I_j is the incoming current.

In discrete time, using timesteps of δt, the neuron is simulated as:

$$V_j(t+\delta t) = \alpha V_j(t) + (1-\alpha) R_m I_j(t) - v_{\text{th}} z_j(t), \tag{D.2}$$

where $\alpha = e^{\left(-\frac{\delta t}{\tau_m}\right)}$, τ_m is the membrane constant of the neuron j. The neuron spike is defined as $z_j(t) = H(V_j(t) - A_j(t))$, where $H(x) = 1$ if $x > 0$ and 0 otherwise, $A_j(t)$ is the adaptive threshold defined below. During the refractory period, $z_j(t)$ is fixed to 0.

In all simulations the R_m was set to 1 GΩ. The input current $I_j(t)$ is defined as the weighted sum of spikes from external inputs and other neurons in the network:

$$I_j(t) = \sum_i W_{ji}^{in} x_i(t - d_{ji}^{in}) + \sum_i W_{ji}^{rec} z_i(t - d_{ji}^{rec}), \tag{D.3}$$

where W_{ji}^{in} and W_{ji}^{rec} denote respectively the input and the recurrent synaptic weights and d_{ji}^{in} and d_{ji}^{rec} the corresponding synaptic delays.

After each spike, the adaptive threshold $A_j(t)$ contains a variable component $a_j(t)$ that is increased by a constant and is allowed to decay back to 0:

$$A_j(t) = v_{th} + \beta a_j(t), a_j(t+\delta t) = \rho_j a_j(t) + (1-\rho_j) z_j(t) \tag{D.4}$$

where v_{th} is the constant baseline of the firing threshold $A(t)$, $a(t)$ is its activity-dependent component, $\beta > 0$ is the relative amplitude of the activity-dependent component. The parameter $\rho = e^{\left(\frac{-\delta t}{\tau_a}\right)}$ controls the speed by which $a(t)$ decays back to 0, where τ_a is the adaptation time constant and δt is the duration of a discrete time step.

We refer to these neurons as adaptive leaky integrate-and-fire (ALIF) neurons. An LSNN is a network of spiking neurons that contains some ALIF neurons.

D.2 Training methods

Since the output of spiking neurons are not differentiable, we use a pseudo-derivative with an additional factor $\gamma < 1$ that dampens the increase of backpropagated errors through spikes as in Bellec et al., 2018b:

$$\frac{dz_j(t)}{dv_j(t)} := \gamma \max\{0, 1 - |v_j(t)|\}, \tag{D.5}$$

where $v_j(t)$ denotes the normalized membrane potential $v_j(t) = \frac{V_j(t) - A_j(t)}{A_j(t)}$.

In ALIF neurons, gradients can be propagated through many time steps through the dynamic threshold without dampening.

D.3 Tasks

Learning priors with the sinusoidal task.

Task family: The LSNN was trained to implement a regression algorithm on a family of sinusoidal functions. The targets were defined by sinusoidal functions $y = A\sin(\phi + x)$ over the domain $x \in [-5, 5]$. The specific function to be learned was defined then by the phase ϕ and the amplitude A, which were chosen uniformly random between $[0, \pi]$ and $[0.1, 5]$ respectively.

Input encoding: Analog values were transformed into spiking trains to serve as inputs to the LSNN as follows: For each input component, 100 input neurons are assigned values $m_1, \ldots m_{100}$ evenly distributed between the minimum and maximum possible value of the input. Each input neuron has a Gaussian response field with a particular mean and standard deviation, where the means are uniformly distributed between the minimum and maximum values to be encoded, and with a constant standard deviation. More precisely, the firing rate r_i (in Hz) of each

input neuron i is given by $r_i = r_{\max} \exp\left(-\frac{(m_i - z_i)^2}{2\sigma^2}\right)$, where $r_{\max} = 200$ Hz, m_i is the value assigned to that neuron, z_i is the analog value to be encoded, and $\sigma = \frac{(m_{\max} - m_{\min})}{1000}$, $m_{\min}$ with $m_{\max}$ being the minimum and maximum values to be encoded.

Output decoding: The output of the LSNN was a linear readout that received as input the mean firing rate of each of the neurons per step i.e the number of spikes divided by 20 for the 20 ms time window that constitutes the step.

LSNN setup and training schedule: The standard LSNN model was used, with 100 hidden neurons. Of these, 40% were adaptive. We used all-to-all connectivity between all neurons (regular and adaptive).

The network training proceeded as follows: A new target function was randomly chosen for each episode of training, i.e., the parameters of the target function were chosen uniformly randomly from within the ranges above. Each episode consisted of a sequence of 500 steps, each lasting for 20 ms. In each step, one training example from the current function to be learned was presented to the LSNN. In such a step, the inputs to the LSNN consisted of a randomly chosen scalar input x. In addition, at each step, the LSNN also got the target value $C(x')$ from the previous step, i.e., the value of the target calculated using the target function for the inputs given at the previous step (in the first step, $C(x')$ is set to 0).

All the weights of the LSNN were updated once per iteration using our variant of BPTT where an iteration consisted of a batch of 100 episodes, and the weight updates were accumulated across episodes in an iteration. Adam Kingma and Ba, 2014 was used with standard parameters and a learning rate of 0.001. The loss function for training was the mean squared error (MSE) of the LSNN predictions over an iteration (i.e. over all the steps in an episode, and over the entire batch of episodes in an iteration). In addition, a regularization term was used to maintain a firing rate of 20 Hz. In this way, we induce the LSNN to use sparse firing. We trained the LSNN for 5000 iterations.

Parameter values: The LSNN parameters were as follows: 5 ms neuronal refractory period, delays of 1 ms, adaptation time constants of the adaptive neurons spread uniformly between $1 - 3000$ ms, $\beta = 1.6$ for adaptive neurons (0 for regular neurons), membrane time constant $\tau = 20$ ms, 0.03 V baseline threshold voltage. The dampening factor for training was $\gamma = 0.3$.

Analysis and comparison: The linear baseline was calculated by performing linear regression on the analog values of input points and targets in the first half of the episodes (250 steps) and testing it on the points in the second half of the episode. The total test MSE was 0.1968 ± 0.1469 and the linear baseline was 4.0340.

Learning to learn motor prediction.

Task family: The family of functions was defined by different two-link arms where the length and masses of the links were randomly chosen in the range $[0.5, 2]$. The torques were generated randomly as described in Gilra and Gerstner, 2017. The network was trained to predict the arm state, i.e. the angles ϕ_1, ϕ_2 of its two links.

Input encoding: The input was encoded in the same way as in the sinusoidal task.

Output decoding: The output of the LSNN was a linear readout that received as input the trace of the firing of all the neurons in the network, where the spiking activity of the neurons was convolved with an exponential kernel with time constant 50 ms to generate the trace.

LSNN setup and training schedule: The standard LSNN model was used, with 600 hidden neurons. Of these, 50% were adaptive in all simulations. We used all-to-all connectivity between all neurons (regular and adaptive).

The training was as follows: During inner loop training, for each episode, we randomly chose a value for the mass and length for each link of the arm. The LSNN received the motor command $m(t) = [m_1(t)m_2(t)]^T$, and the actual state vector of the arm $\tau = 100$ ms ago, $s(t-\tau)$ as inputs. The state vector of the arm $s(t) = [\phi_1(t)\phi_2(t)]^T$ was defined by the angles ϕ_1, ϕ_2 of its two links. All the inputs were encoded into spikes using a population-rate code before being presented to the network (as shown in Fig. 4.3C top panel). A linear readout on the trace of the neural activity was used to generate the predictions of the state of the arm $\hat{s}(t;\Theta)$. Each episode lasted for 10 seconds, where the torque changed every 5 ms.

In the outerloop, the following loss function was minimized using BPTT for spiking networks:

$$\mathcal{L}(\Theta) = \mathbb{E}_{C\sim\mathcal{F}}\left[\int_t \left(\boldsymbol{s}(t) - \widehat{\boldsymbol{s}}(t;\Theta)\right)^2\right], \tag{D.6}$$

where $\mathbb{E}$ denotes expectation, and Θ denotes the hyper-parameters.

Parameter values: The LSNN parameters were as follows: 5 ms neuronal refractory period, delays of 1 ms, adaptation time constants of the adaptive neurons spread uniformly between $1-600$ ms, $\beta = 1.7$ for adaptive neurons (0 for regular neurons), membrane time constant $\tau = 20$ ms, 0.03 mV baseline threshold voltage. The dampening factor for training was $\gamma = 0.3$. We used the Adam optimizer Kingma and Ba, 2014 with the default parameters with a learning rate of 0.001. We used a batch size of 80 for training.

Meta-reinforcement learning.

Task family: An LSNN-based agent was trained on a family of navigation tasks in a two-dimensional circular arena. For all tasks, the arena is a circle with radius 1 and goals are smaller circles of radius 0.3 with centers uniformly distributed on

the circle of radius 0.85. At the beginning of an episode and after the agent reaches a goal, the agent's position is set randomly with uniform probability within the arena. At every timestep, the agent chooses an action by generating a small velocity vector of Euclidean norm smaller or equal to $a_{scale} = 0.02$. When the agent reaches the goal, it receives a reward of 1. If the agent attempts to move outside the arena, the new position is given by the intersection of the velocity vector with the border and the agent receives a negative reward of -0.02.

Input encoding: Information of the current environmental state $s(t)$ and the reward $r(t)$ were provided to the LSNN at each time step t as follows: The state $s(t)$ is given by the x and y coordinate of the agent's position (see top of Fig. 4.4C). Each position coordinate $\xi(t) \in [-1, 1]$ is encoded by 40 neurons which spike according to a Gaussian population rate code defined as follows: A preferred coordinate value ξ_i, is assigned to each of the 40 neurons, where ξ_i's are evenly spaced between -1 and 1. The firing rate of neuron i is then given by $r_{\max} \exp(-100(\xi_i - \xi)^2)$ where $r_{\max}$ is 500 Hz. The instantaneous reward $r(t)$ is encoded by two groups of 40 neurons (see green row at the top of Fig. 4.4F). All neurons in the first group spike in synchrony each time a reward of 1 is received (i.e., the goal was reached), and the ones in the second group spike when a reward of -0.02 is received (i.e., the agent moved into a wall).

Output decoding: The output of the LSNN is provided by five readout neurons. Their membrane potentials $y_i(t)$ define the outputs of the LSNN. The action vector $\boldsymbol{a}(t) = (a_x(t), a_y(t))^T$ is sampled from the distribution $\pi_{\boldsymbol{\theta}}$ which depends on the network parameters $\boldsymbol{\theta}$ through the readouts $y_i(t)$ as follows: The coordinate $a_x(t)$ ($a_y(t)$) is sampled from a Gaussian distribution with mean $\mu_x = \tanh(y_1(t))$ ($\mu_y = \tanh(y_2(t))$) and variance $\phi_x = \sigma(y_3(t))$ ($\phi_y = \sigma(y_4(t))$). The velocity vector that updates the agent's position is then defined as $a_{scale}\, \boldsymbol{a}(t)$. If this velocity has a norm larger than a_{scale}, it is clipped to a norm of a_{scale}.

The last readout output $y_5(t)$ is used to predict the value function $V_{\boldsymbol{\theta}}(t)$. It estimates the expected discounted sum of future rewards $R(t) = \sum_{t'>t} \eta^{t'-t} r(t')$, where $\eta = 0.99$ is the discount factor and $r(t')$ denotes the reward at time t'. To enable the network to learn complex forms of exploration we introduced current noise in the neuron model in this task. At each time step, we added a small Gaussian noise with mean 0 and standard deviation $\frac{1}{R_m}\nu_j$ to the current I_j into neuron j. Here, ν_j is a network parameter initialized at 0.03 and optimized by BPTT alongside the network weights.

LSNN setup and training schedule: To train the network we used the Proximal Policy Optimization algorithm (PPO) Schulman et al., 2017. For each training iteration, K full episodes of T timesteps were generated with fixed parameters $\boldsymbol{\theta}_{old}$ (here $K = 10$ and $T = 2000$). We write the clipped surrogate objective of PPO as $O^{PPO}(\boldsymbol{\theta}_{old}, \boldsymbol{\theta}, t, k)$ (this is defined under the notation L^{CLIP} in Schulman et al., 2017). The loss with respect to $\boldsymbol{\theta}$ is then defined as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{KT}\sum_{k<K}\sum_{t<T} O^{PPO}(\boldsymbol{\theta}_{old}, \boldsymbol{\theta}, t, k) + \mu_v \left(R(t,k) - V_{\boldsymbol{\theta}}(t,k)\right)^2, \quad \text{(D.7)}$$

$$-\mu_e H(\pi_{\boldsymbol{\theta}}(k,t)) + \mu_{firing}\frac{1}{n}\sum_j ||\frac{1}{KT}\sum_{k,t} z_j(t,k) - f^0||^2, \quad \text{(D.8)}$$

where $H(\pi_{\boldsymbol{\theta}})$ is the entropy of the distribution $\pi_{\boldsymbol{\theta}}$, f^0 is a target firing rate of 10 Hz, and μ_v, μ_e, μ_{firing} are regularization hyper-parameters. Importantly, probability distributions used in the definition of the loss $\mathcal{L}$ (i.e. the trajectories) are conditioned on the current noises, so that for the same noise and infinitely small parameter change from $\boldsymbol{\theta}_{old}$ to $\boldsymbol{\theta}$ the trajectories and the spike trains are the same. At each iteration this loss function $\mathcal{L}$ is then minimized with one step of the ADAM optimizer Kingma and Ba, 2014.

Parameter values: In this task the LSNN had 400 hidden units (200 excitatory neurons, 80 inhibitory neurons and 120 adaptive neurons with adaptation time constants $\tau_a = 1200$ ms) and the network was rewired with a fixed global connectivity of 20% Bellec et al., 2018a. The membrane time constants were similarly sampled between 15 and 30 ms. The adaptation amplitude β was set to 1.7. The refractory period was set to 3 ms and delays were sampled uniformly between 1 and 10 ms. The regularization parameters μ_v, μ_e and μ_{firing} were respectively 1, 0.001, and 100. The parameter ϵ of the PPO algorithm was set to 0.2. The learning rate was initialized to 0.01 and decayed by a factor 0.5 every 5000 iterations. We used the default parameters for ADAM, except for the parameter ϵ which we set to 10^{-5}.

Appendix E

Appendix to Chapter 5: Slow processes of neurons enable a biologically plausible approximation to policy gradient

E.1 Detailed derivation of *Reward-based e-prop* for continuous actions

Here, we show the detailed derivation for the synaptic plasticity rules that result from gradients of the loss function E, as given in equation (5.1), see Fig. E.1 for the network architecture. As a result of the general actor-critic framework with policy gradient, this loss function additively combines the loss function for the policy E_π (actor) and the value function E_V (critic).

We assume that the agent can take at certain decision times $t_0, \ldots, t_n, \ldots$ real-valued actions $\boldsymbol{a}$. We also assume that each component k of this action vector follows independent Gaussian distributions, with a mean given by the output y_k and a fixed variance σ^2.

We consider first the regression problem defined by the loss function E_V. We obtain an estimation of the loss gradient:

$$\widehat{\frac{dE_V}{dW_{ji}^{\text{rec}}}} = -\sum_{t'} (R^{t'} - V^{t'}) W_j^{V,\text{out}} \, \bar{e}_{ji}^{t'} \,, \tag{E.1}$$

where $W_j^{V,\text{out}}$ are the weights of the output neuron V_j^t predicting the value function $\mathbb{E}[R^t]$. In order to overcome the obstacle that an evaluation of the return $R^{t'}$ requires to know future rewards, we introduce temporal difference errors $\delta^t = r^t + \gamma V^{t+1} - V^t$, and use that $R^{t'} - V^{t'}$ is equal to the sum $\sum_{t \geq t'} \gamma^{t-t'} \delta^t$. We then reorganize the two sums over t and t' (note that the interchange of the summation order amounts to the equivalence between forward and backward view of RL (R. S.

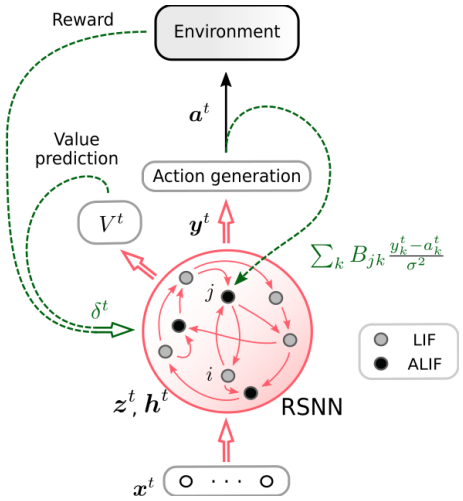

Figure E.1: Learning architecture for *reward-based e-prop*: The network input $\mathbf{x}^t$ consists of the current joint angles and input cues. The network produces output $\mathbf{y}^t$ which is used to stochastically generate the actions $\mathbf{a}^t$. In addition, the network produces the value prediction, which, along with the reward from the environment, is used to calculate the TD-error δ^t. The learning signals and the TD-errors are used to calculate the weight update, as denoted by the green dotted lines.

Sutton and Barto, 2018)):

$$\widehat{\frac{dE_V}{dW_{ji}^{\text{rec}}}} = -\sum_{t'} \Big(\sum_{t \geq t'} \gamma^{t-t'} \delta^t \Big) W_j^{V,\text{out}} \, \bar{e}_{ji}^{t'} \tag{E.2}$$

$$= -\sum_{t} \delta^t \sum_{t' \leq t} \gamma^{t-t'} W_j^{V,\text{out}} \, \bar{e}_{ji}^{t'} \tag{E.3}$$

$$= -\sum_{t} \delta^t \, \mathcal{F}_\gamma \big(W_j^{V,\text{out}} \, \bar{e}_{ji}^{t} \big) \, . \tag{E.4}$$

For the other part E_π in the loss function E defined at equation (5.1), we consider the estimator $\widehat{\frac{\partial \log \pi(\boldsymbol{a}^t | \boldsymbol{y}^t)}{\partial z_j^t}}$, and use our previous definition that each component k of the action follows an independent Gaussian, which has a mean given by the output y_k and a fixed variance σ^2. The estimator then becomes:

$$\widehat{\frac{\partial \log \pi(\boldsymbol{a}^t | \boldsymbol{y}^t)}{\partial z_j^t}} = -\sum_k W_{kj}^{\pi,\text{out}} \sum_{\{n \,|\, t_n \geq t\}} \kappa^{t_n - t} (R^{t_n} - V^{t_n}) \frac{a_k^{t_n} - y_k^{t_n}}{\sigma^2} \, , \tag{E.5}$$

where $W_{kj}^{\pi,\text{out}}$ are the weights onto the output neurons y_k^t defining the policy π, and κ is the constant of the low-pass filtering of the output neurons. Using e-prop, we

arrive at an estimation of the loss gradient of the form:

$$\widehat{\frac{dE_\pi}{dW_{ji}^{\text{rec}}}} = \sum_t \widehat{\frac{\partial E_\pi}{\partial z_j^t}} e_{ji}^t \tag{E.6}$$

$$= -\sum_{t,k} W_{kj}^{\pi,\text{out}} \sum_{\{n \,|\, t_n \geq t\}} (R^{t_n} - V^{t_n}) \frac{a_k^{t_n} - y_k^{t_n}}{\sigma^2} \kappa^{t_n - t} e_{ji}^t \tag{E.7}$$

$$= -\sum_{n,k} (R^{t_n} - V^{t_n}) W_{kj}^{\pi,\text{out}} \frac{a_k^{t_n} - y_k^{t_n}}{\sigma^2} \underbrace{\sum_{t \leq t_n} \kappa^{t_n - t} e_{ji}^t}_{\bar{e}_{ji}^{t_n}} . \tag{E.8}$$

Like in the derivation of the gradient of E_V, this formula hides a sum over future rewards in R^{t_n} that cannot be computed online. It is resolved by introducing the backward view as in equation (E.4). We arrive at the loss gradient:

$$\widehat{\frac{dE_\pi}{dW_{ji}^{\text{rec}}}} = -\sum_t \delta^t \mathcal{F}_\gamma \Big(\sum_k W_{kj}^{\pi,\text{out}} \frac{a_k^t - y_k^t}{\sigma^2} \, \bar{e}_{ji}^t \Big) . \tag{E.9}$$

Importantly, an action is only taken at times $t_0, \ldots, t_n, \ldots$, hence for all other times, we set the term $(a_k^t - y_k^t)$ to zero.

Finally, the gradient of the loss function E is the sum of the gradients of E_π and E_V, equations (E.4) and (E.9) respectively. Application of stochastic gradient descent with a learning rate of η yields the synaptic plasticity rule given in the equations (5.6) and (5.7).

The gradient of E with respect to the output weights can be computed directly from equation (5.1) without the theory of *e-prop*. However, it also needs to account for the sum over future rewards that is present in the term $R^t - V^t$. Using a similar derivation as in equations (E.2)-(E.4) the plasticity rule for these weights becomes:

$$\Delta W_{kj}^{\pi,\text{out}} = -\eta \sum_t \delta^t \mathcal{F}_\gamma \Big(\frac{y_k^t - a_k^t}{\sigma^2} \mathcal{F}_\kappa(z_j^t) \Big), \tag{E.10}$$

$$\Delta W_j^{\text{V,out}} = \eta c_V \sum_t \delta^t \mathcal{F}_\gamma \Big(\mathcal{F}_\kappa(z_j^t) \Big) . \tag{E.11}$$

Similarly, we also obtain for the update rules of the biases of the output neurons: $\Delta b_k^{\pi,\text{out}} = -\eta \sum_t \delta^t \mathcal{F}_\gamma \Big(\frac{y_k^t - a_k^t}{\sigma^2} \Big)$, and $\Delta b^{\text{V,out}} = \eta c_V \sum_t \delta^t$.

E.2 Simulation details: delayed arm reaching task

Details of the arm model: The arm consisted of two links, with one link connected to the other link by a joint, which is itself connected by a joint to a fixed position in space. The configuration of this arm model at time t can be described by the

angles ϕ_1^t and ϕ_2^t of the two joints measured against the horizontal and the first link of the arm respectively, see Fig. 5.1C. For given angles, the position $\boldsymbol{y}^t = (x^t, y^t)$ of the tip of the arm in Euclidean space is given by $x^t = l\cos(\phi_1^t) + l\cos(\phi_1^t + \phi_2^t)$ and $y^t = l\sin(\phi_1^t) + l\sin(\phi_1^t + \phi_2^t)$. Angles were computed by discrete integration over time: $\phi_i^t = \sum_{t' \leq t} \dot{\phi}_i^{t'} \delta t + \phi_i^0$ using $\delta t = 1\,\text{ms}$.

Details of the delayed arm reaching task and of the input scheme: The agent could control the arm by setting the angular velocities of the two joints to a different value at every ms. There was a total of 8 possible goal locations, which were evenly distributed on a circle with a radius of 0.8. The arm was initially positioned so that its tip was equidistant from all the goals. In each trial, one of the 8 goals was chosen randomly, and indicated as the desired goal location in the first 100 ms of the trial. Each possible goal location was associated with a separate input channel, consisting of 20 neurons. They produced a Poisson spike train with a rate of 500 Hz while the corresponding goal location was indicated. After this cue was provided, a delay period of a randomly chosen length between $100 - 500$ ms started, during which the subject was penalized with a negative reward of -0.1 if it moved outside a central region of radius 0.3. After this delay period, a go cue instructed the subject to move towards the goal location. This cue was provided in a separate input channel of 20 neurons, which produced a Poisson spike train with a rate of 500 Hz for 100 ms. Once the tip of the arm had moved closer than a distance of 0.1 to the goal location, a positive reward of 1 was given to signal a success. A negative reward of -0.01 was given for every ms after the go cue started while the arm did not yet reach the goal, in order to encourage an efficient movement. Going far off the region of interest — a circle of radius 1 — was penalized with a negative reward of -0.1 at each ms. One trial lasted for a total of 1.5 seconds i.e. the subject had 900 ms from the start of the go cue to reach the goal.

The agent also received its current configuration (angles of the arms ϕ_1 and ϕ_2, see Fig. 5.1B) as input at each time step in the following way: Each one of the angles was encoded by a population of 30 neurons, where each neuron had a Gaussian tuning curve centered on values distributed evenly between 0 and 2π, with a firing rate peak of 100 Hz. The tuning curve had a standard deviation of $\frac{4}{30}$.

In addition, if the goal position was successfully reached, the network received this information using a separate input channel consisting of 20 neurons that produced a Poisson spike train with a rate of 500 Hz.

Details of the network model: The network consisted of 350 LIF neurons and 150 ALIF neurons. The membrane time constant of all neurons was $\tau_m = 20$ ms, with a baseline threshold $v_{\text{th}} = 0.6$ and a refractory period of 3 ms. All synaptic delays were 1 ms. The adaptation time constant of ALIF neurons was set to $\tau_a = 500$ ms, and the adaptation strength was $\beta_j = 0.07$. The membrane time constant of output neurons was given by $\tau_{out} = 20$ ms.

Actions (angular velocities for the 2 joints) were sampled from a Gaussian distribution with a mean of y_k^t, and a standard deviation of $\sigma = 0.1$, which was exponentially decayed over iterations so that it reached $\sigma = 0.01$ at the end.

Details of the learning procedure: The network was trained for a total of 16000 weight updates (iterations). In each iteration, a batch of 200 trials was simulated, and we applied weight changes at the end of each batch. Independent of the learning method, we used Adam to implement the weight update, with a learning rate of 0.001 and default hyper-parameters (Kingma and Ba, 2014). For training with *BPTT*, gradients were computed for the loss function given in equation (5.1). In the case of *e-prop*, we used equations (5.6) and (5.7). For *random e-prop*, the broadcast weights B_{jk} were initialized using a Gaussian distribution with mean 0 and variance 1. To avoid an excessively high firing rate, regularization was applied with $c_{\text{reg}} = 100$ and a target firing rate of $f^{\text{target}} = 10$ Hz.

Bibliography

Abraham, W. C. and M. F. Bear (1996). "Metaplasticity: the plasticity of synaptic plasticity". In: *Trends in neurosciences* 19.4, pp. 126–130 (cit. on p. 30).

Allen Institute (Oct. 2017). *Allen Cell Types Database Technical white paper: GLIF models*. Tech. rep. v4 (cit. on p. 9).

Allen Institute (2018). "© 2018 Allen Institute for Brain Science. Allen Cell Types Database, cell feature search. Available from: `celltypes.brain-map.org/data`". In: (cit. on pp. 8–10, 24, 67, 68).

Bailey, C. H. and M. Chen (1988). "Morphological basis of short-term habituation in Aplysia". In: *Journal of Neuroscience* 8.7, pp. 2452–2459 (cit. on p. 48).

Barbosa, J., H. Stein, R. Martinez, A. Galan, K. Adam, S. Li, J. Valls-Solé, C. Constantinidis, and A. Compte (2019). "Interplay between persistent activity and activity-silent dynamics in prefrontal cortex during working memory." In: *bioRxiv*, p. 763938 (cit. on p. 26).

Barnett, M. W. and P. M. Larkman (2007). "The action potential". In: *Practical neurology* 7.3, pp. 192–197 (cit. on p. 4).

Barone, P. and J.-P. Joseph (1989). "Prefrontal cortex and spatial sequencing in macaque monkey". In: *Experimental brain research* 78.3, pp. 447–464 (cit. on p. 21).

Behrens, T. E., M. W. Woolrich, M. E. Walton, and M. F. Rushworth (2007). "Learning the value of information in an uncertain world". In: *Nature neuroscience* 10.9, pp. 1214–1221 (cit. on p. 42).

Bellec, G., D. Kappel, W. Maass, and R. Legenstein (2018a). "Deep Rewiring: Training very sparse deep networks". In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 18, 53, 90).

Bellec, G., D. Salaj, A. Subramoney, R. Legenstein, and W. Maass (2018b). "Long short-term memory and Learning-to-learn in networks of spiking neurons". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., pp. 795–805 (cit. on pp. 1, 3, 5, 11, 31, 39, 40, 43, 47, 58, 60, 69, 79, 85, 86).

Bellec, G., F. Scherr, E. Hajek, D. Salaj, R. Legenstein, and W. Maass (Jan. 2019a). "Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets". In: *arXiv:1901.09049 [cs]*. arXiv: 1901.09049 (cit. on p. 40).

Bellec, G., F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass (Aug. 2019b). "A solution to the learning dilemma for recurrent networks of spiking neurons". en. In: *bioRxiv* (cit. on pp. 2, 11, 21, 40, 56, 70).

Bohnstingl, T., F. Scherr, C. Pehle, K. Meier, and W. Maass (2019). "Neuromorphic Hardware Learns to Learn". In: *Frontiers in Neuroscience* 13, p. 483 (cit. on p. 40).

Botvinick, M., S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis (May 2019). "Reinforcement Learning, Fast and Slow". English. In: *Trends in Cognitive Sciences* 23.5, pp. 408–422 (cit. on pp. 42, 43).

Carpenter, A. F., G. Baud-Bovy, A. P. Georgopoulos, and G. Pellizzer (2018). "Encoding of serial order in working memory: neuronal activity in motor, premotor, and prefrontal cortex during a memory scanning task". In: *Journal of Neuroscience* 38.21, pp. 4912–4933 (cit. on pp. 21, 25).

Cassenaer, S. and G. Laurent (2012). "Conditional modulation of spike-timing-dependent plasticity for olfactory learning". In: *Nature* 482.7383, p. 47 (cit. on p. 56).

Chettih, S. N. and C. D. Harvey (2019). "Single-neuron perturbations reveal feature-specific competition in V1". In: *Nature* 567.7748, pp. 334–340 (cit. on p. 24).

Cotter, N. E. and P. R. Conwell (June 1990). "Fixed-weight networks can learn". In: *1990 IJCNN International Joint Conference on Neural Networks*, 553–559 vol.3 (cit. on pp. 3, 43).

Courbariaux, M., I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio (2016). "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1". In: *arXiv preprint arXiv:1602.02830* (cit. on p. 79).

Cramer, B., Y. Stradmann, J. Schemmel, and F. Zenke (2019). "The Heidelberg spiking datasets for the systematic evaluation of spiking neural networks". In: *arXiv preprint arXiv:1910.07407* (cit. on p. 17).

Duan, Y., J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel (2016). "RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *arXiv preprint arXiv:1611.02779* (cit. on pp. 38, 43, 51).

Engelhard, B., J. Finkelstein, J. Cox, W. Fleming, H. J. Jang, S. Ornelas, S. A. Koay, S. Y. Thiberge, N. D. Daw, D. W. Tank, et al. (2019). "Specialized coding of sensory, motor and cognitive variables in VTA dopamine neurons". In: *Nature*, p. 1 (cit. on p. 56).

Esser, S. K., P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. d. Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha (Nov. 2016). "Convolutional networks for fast, energy-efficient neuromorphic computing". en. In: *Proceedings of the National Academy of Sciences* 113.41, pp. 11441–11446 (cit. on p. 79).

Field, A. (2013). *Discovering statistics using IBM SPSS statistics*. sage (cit. on pp. 25, 76).

Froemke, R. C., D. Debanne, and G.-Q. Bi (2010). "Temporal modulation of spike-timing-dependent plasticity". In: *Frontiers in synaptic neuroscience* 2, p. 19 (cit. on p. 42).

Gerstner, W. and W. M. Kistler (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press (cit. on p. 4).

Gerstner, W., W. M. Kistler, R. Naud, and L. Paninski (2014). *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press (cit. on p. 9).

Gerstner, W., M. Lehmann, V. Liakoni, D. Corneil, and J. Brea (July 2018). "Eligibility Traces and Plasticity on Behavioral Time Scales: Experimental Support of NeoHebbian Three-Factor Learning Rules". In: *Frontiers in Neural Circuits* 12 (cit. on pp. 56, 62).

Gilra, A. and W. Gerstner (2017). "Predicting non-linear dynamics by stable local learning in a recurrent spiking neural network". In: *Elife* 6, e28295 (cit. on pp. 51, 88).

Glorot, X. and Y. Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256 (cit. on pp. 80, 84).

Gutkin, B. and F. Zeldenrust (2014). "Spike frequency adaptation". In: *Scholarpedia* 9.2. revision #143322, p. 30643 (cit. on p. 8).

Haeusler, S. and W. Maass (2006). "A statistical analysis of information-processing properties of lamina-specific cortical microcircuit models". In: *Cerebral cortex* 17.1, pp. 149–162 (cit. on p. 30).

Harlow, H. F. (1949). "The formation of learning sets." In: *Psychological review* 56.1, p. 51 (cit. on pp. 3, 42).

Harvey, C. D., P. Coen, and D. W. Tank (2012). "Choice-specific sequences in parietal cortex during a virtual-navigation decision task". In: *Nature* 484.7392, pp. 62–68 (cit. on p. 24).

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780 (cit. on pp. 43, 56).

Hochreiter, S., A. S. Younger, and P. R. Conwell (2001). "Learning to learn using gradient descent". In: *International Conference on Artificial Neural Networks*. Springer, pp. 87–94 (cit. on pp. 3, 30, 31, 35, 38, 43, 45, 47).

Hodgkin, A. L., A. F. Huxley, and B. Katz (1952). "Measurement of current-voltage relations in the membrane of the giant axon of Loligo". In: *The Journal of physiology* 116.4, pp. 424–448 (cit. on p. 4).

Hu, B., M. E. Garrett, P. A. Groblewski, D. R. Ollerenshaw, J. Shang, K. Roll, S. Manavi, C. Koch, S. R. Olsen, and S. Mihalas (2020). "Adaptation supports short-term memory in a visual change detection task." In: *bioRxiv* (cit. on pp. 26, 27).

Huh, D. and T. J. Sejnowski (2018). "Gradient descent for spiking neural networks". In: *Advances in Neural Information Processing Systems*, pp. 1433–1443 (cit. on pp. 16–18).

Inagaki, H. K., L. Fontolan, S. Romani, and K. Svoboda (2019). "Discrete attractor dynamics underlies persistent activity in the frontal cortex." In: *Nature* 566.7743, pp. 212–217 (cit. on p. 25).

Jaeger, H. (2001). "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note". In: *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148, p. 34 (cit. on pp. 30, 44).

Kamiński, J. and U. Rutishauser (2019). "Between persistently active and activity-silent frameworks: novel vistas on the cellular basis of working memory". In: *Annals of the New York Academy of Sciences* (cit. on pp. 25, 26).

Kappel, D., R. Legenstein, S. Habenschuss, M. Hsieh, and W. Maass (2018). "A dynamic connectome supports the emergence of stable computational function of neural circuits through reward-based learning". In: *eNeuro* (cit. on p. 62).

Kingma, D. P. and J. Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (cit. on pp. 80, 83, 84, 87, 88, 90, 95).

Kleim, J. A., T. M. Hogg, P. M. VandenBerg, N. R. Cooper, R. Bruneau, and M. Remple (2004). "Cortical synaptogenesis and motor map reorganization occur during late, but not early, phase of motor skill learning". In: *Journal of Neuroscience* 24.3, pp. 628–633 (cit. on p. 48).

Kullmann, D. M., A. W. Moreau, Y. Bakiri, and E. Nicholson (2012). "Plasticity of inhibition". In: *Neuron* 75.6, pp. 951–962 (cit. on p. 68).

Kusupati, A., M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma (2018). "Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network". In: *Advances in Neural Information Processing Systems*, pp. 9017–9028 (cit. on p. 17).

Lalazar, H. and E. Vaadia (2008). "Neural basis of sensorimotor learning: modifying internal models". In: *Current opinion in neurobiology* 18.6, pp. 573–581 (cit. on p. 48).

Lashley, K. S. (1951). *The problem of serial order in behavior*. Vol. 21. Bobbs-Merrill Oxford, United Kingdom (cit. on p. 21).

Levenstein, D., V. A. Alvarez, A. Amarasingham, H. Azab, R. C. Gerkin, A. Hasenstaub, R. Iyer, R. B. Jolivet, S. Marzen, J. D. Monaco, A. A. Prinz, S. Quraishi, F. Santamaria, S. Shivkumar, M. F. Singh, D. B. Stockton, R. Traub, H. G. Rotstein, F. Nadim, and A. D. Redish (Apr. 2020). "On the role of theory and modeling in neuroscience". In: *arXiv:2003.13825 [q-bio]* (cit. on p. 1).

Lillicrap, T. P. and A. Santoro (2019). "Backpropagation through time and the brain". In: *Current Opinion in Neurobiology* 55, pp. 82–89 (cit. on p. 56).

Lillicrap, T. P., D. Cownden, D. B. Tweed, and C. J. Akerman (2016). "Random synaptic feedback weights support error backpropagation for deep learning". In: *Nature Communications* 7, p. 13276 (cit. on p. 59).

Lindsay, G. W., M. Rigotti, M. R. Warden, E. K. Miller, and S. Fusi (2017). "Hebbian learning in a random network captures selectivity properties of the prefrontal cortex". In: *Journal of Neuroscience* 37.45, pp. 11021–11036 (cit. on p. 75).

Liu, Y., R. J. Dolan, Z. Kurth-Nelson, and B. T. E. (2019). "Human replay spontaneously reorganizes experience". In: *Cell* 178.3, pp. 640–652 (cit. on pp. 21, 22).

Maass, W., T. Natschläger, and H. Markram (Nov. 2002). "Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations". In: *Neural Computation* 14.11, pp. 2531–2560 (cit. on pp. 3, 30, 44).

Maass, W. (1997). "Networks of spiking neurons: the third generation of neural network models". In: *Neural networks* 10.9, pp. 1659–1671 (cit. on pp. 1, 4).

Maass, W. and H. Markram (Dec. 2004). "On the computational power of circuits of spiking neurons". In: *Journal of Computer and System Sciences* 69.4, pp. 593–616 (cit. on p. 1).

Maass, W., T. Natschläger, and H. Markram (2004). "Fading memory and kernel properties of generic cortical microcircuit models". In: *Journal of Physiology-Paris* 98.4-6, pp. 315–330 (cit. on pp. 3, 30).

MacDonald III, A. W. (2008). "Building a clinically relevant cognitive task: case study of the AX paradigm". In: *Schizophrenia bulletin* 34.4, pp. 619–628 (cit. on pp. 2, 20).

MacLean, S. J., C. D. Hassall, Y. Ishigami, O. E. Krigolson, and G. A. Eskes (2015). "Using brain potentials to understand prism adaptation: the error-related negativity and the P300". In: *Frontiers in human neuroscience* 9, p. 335 (cit. on p. 56).

Marcus, G. F. (2003). *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. MIT Press (cit. on pp. 2, 21, 22).

Markram, H., E. Muller, S. Ramaswamy, M. W. Reimann, M. Abdellah, C. A. . Sanchez, and G. A. A. Kahou (2015). "Reconstruction and simulation of neocortical microcircuitry." In: *Cell* 163.2, pp. 456–492 (cit. on p. 15).

Masse, N. Y., G. R. Yang, H. F. Song, X.-J. Wang, and D. J. Freedman (2019). "Circuit mechanisms for the maintenance and manipulation of information in working memory". In: *Nature Neuroscience*, p. 1 (cit. on pp. 25–27).

Mensi, S., R. Naud, C. Pozzorini, M. Avermann, C. C. Petersen, and W. Gerstner (2012). "Parameter extraction and classification of three cortical neuron types reveals two distinct adaptation mechanisms". In: *Journal of neurophysiology* 107.6, pp. 1756–1775 (cit. on p. 68).

Mitchell, T. M. (1980). *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research ... (cit. on p. 1).

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016). "Asynchronous methods for deep reinforcement learning". In: *ICML*, pp. 1928–1937 (cit. on pp. 4, 56).

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (Feb. 2015a). "Human-level control through deep reinforcement learning". en. In: *Nature* 518.7540, pp. 529–533 (cit. on p. 4).

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. (2015b). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, p. 529 (cit. on p. 56).

Mongillo, G., O. Barak, and M. Tsodyks (2008). "Synaptic theory of working memory". In: *Science* 319.5869, pp. 1543–1546 (cit. on pp. 15, 26, 68, 69).

Morris, R. (1984). "Developments of a water-maze procedure for studying spatial learning in the rat". In: *Journal of neuroscience methods* 11.1, pp. 47–60 (cit. on pp. 51, 54).

Nøkland, A. (2016). "Direct feedback alignment provides learning in deep neural networks". In: *NIPS*, pp. 1037–1045 (cit. on p. 59).

O'Reilly, R. C. and M. J. Frank (2006). "Making working memory work: a computational model of learning in the prefrontal cortex and basal ganglia". In: *Neural computation* 18.2, pp. 283–328 (cit. on pp. 2, 18, 20).

Olivers, C. N., J. Peters, R. Houtkamp, and P. R. Roelfsema (2011). "Different states in visual working memory: When it guides attention and when it does not." In: *Trends in cognitive sciences* 15.7, pp. 327–334 (cit. on pp. 25, 26).

Pascanu, R., T. Mikolov, and Y. Bengio (Feb. 2013). "On the difficulty of training recurrent neural networks". en. In: *International Conference on Machine Learning*, pp. 1310–1318 (cit. on p. 1).

Perich, M. G., J. A. Gallego, and L. Miller (Nov. 2018). "A Neural Population Mechanism for Rapid Learning". en. In: *Neuron*, 964–976.e7 (cit. on pp. 35, 43, 44, 50, 54).

Pozzi, I., S. Bohté, and P. Roelfsema (2018). "A biologically plausible learning rule for deep learning in the brain". In: *arXiv preprint arXiv:1811.01768* (cit. on p. 62).

Pozzorini, C., S. Mensi, O. Hagens, R. Naud, C. Koch, and W. Gerstner (2015). "Automated high-throughput characterization of single neurons by means of simplified spiking models". In: *PLoS computational biology* 11.6 (cit. on pp. 9, 26, 68).

Pozzorini, C., R. Naud, S. Mensi, and W. Gerstner (2013). "Temporal whitening by power-law adaptation in neocortical neurons". In: *Nature neuroscience* 16.7, p. 942 (cit. on pp. 9, 26, 68).

Rao, A., A. Subramoney, R. Legenstein, and W. Maass (2020). "Dendritic learning in layer 5 pyramidal cells approximates logistic regression". In: *(In preparation)* (cit. on p. 6).

Reddi, S. J., S. Kale, and S. Kumar (2018). "On the convergence of adam and beyond". In: *International Conference on Learning Representations* (cit. on p. 84).

Roeper, J. (2013). "Dissecting the diversity of midbrain dopamine neurons". In: *Trends in neurosciences* 36.6, pp. 336–342 (cit. on p. 56).

Rougier, N. P., D. C. Noelle, T. S. Braver, J. D. Cohen, and R. C. O'Reilly (May 2005). "Prefrontal cortex and flexible cognitive control: Rules without symbols". en. In: *Proceedings of the National Academy of Sciences* 102.20, pp. 7338–7343 (cit. on p. 2).

Sajad, A., D. C. Godlove, and J. D. Schall (Feb. 2019). "Cortical microcircuitry of performance monitoring". En. In: *Nature Neuroscience* 22.2, p. 265 (cit. on p. 56).

Salaj, D., A. Subramoney, C. Kraisnikovic, G. Bellec, R. Legenstein, and W. Maass (May 2020). "Spike-frequency adaptation provides a long short-term memory to networks of spiking neurons". en. In: *bioRxiv*, p. 2020.05.11.081513 (cit. on pp. 3, 39, 43).

Sanhueza, M. and J. Lisman (2013). "The CaMKII/NMDAR complex as a molecular memory". In: *Molecular brain* 6.1, p. 10 (cit. on p. 56).

Schmidhuber, J. (1987). "Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook". PhD thesis. Technische Universität München (cit. on p. 2).

Schrittwieser, J., I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver (Feb. 2020). "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model". In: *arXiv:1911.08265 [cs, stat]*. 00020 arXiv: 1911.08265 (cit. on p. 4).

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (cit. on pp. 53, 89).

Seijen, H. van and R. Sutton (Jan. 2014). "True Online TD(lambda)". en. In: *International Conference on Machine Learning*. 00063, pp. 692–700 (cit. on p. 60).

Siegelmann, H. T. and E. D. Sontag (Feb. 1995). "On the Computational Power of Neural Nets". In: *Journal of Computer and System Sciences* 50.1, pp. 132–150 (cit. on p. 1).

Silver, D., T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419, pp. 1140–1144 (cit. on p. 56).

Subramoney, A. and W. Maass (2020). "The critical regime enables networks of spiking neurons to maintain a working memory during cognitive computations". In: *(In preparation)* (cit. on p. 6).

Subramoney, A., F. Scherr, and W. Maass (Sept. 2019). "Reservoirs learn to learn". In: *arXiv:1909.07486 [cs]* (cit. on p. 44).

Sutton, R. S. and A. G. Barto (2018). *Reinforcement Learning: An Introduction*. MIT press (cit. on pp. 4, 57, 58, 60, 91).

Tanaka, G., T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose (2019). "Recent advances in physical reservoir computing: A review". In: *Neural Networks* 115, pp. 100–123 (cit. on p. 40).

Tartaglia, E. M., G. Mongillo, and N. Brunel (2015). "On the relationship between persistent delay activity, repetition enhancement and priming". In: *Frontiers in psychology* 5, p. 1590 (cit. on p. 68).

Teeter, C., R. Iyer, V. Menon, N. Gouwens, D. Feng, J. Berg, A. Szafer, N. Cain, H. Zeng, M. Hawrylycz, et al. (2018). "Generalized leaky integrate-and-fire models classify multiple neuron types". In: *Nature communications* 9.1, pp. 1–15 (cit. on p. 9).

Thoroughman, K. A. and R. Shadmehr (2000). "Learning of action through adaptive combination of motor primitives". In: *Nature* 407.6805, p. 742 (cit. on p. 48).

Thrun, S. and L. Pratt (1998). "Learning to Learn: Introduction and Overview". en. In: *Learning to Learn*. Ed. by S. Thrun and L. Pratt. 00107. Boston, MA: Springer US, pp. 3–17 (cit. on p. 2).

Trübutschek, D., S. Marti, A. Ojeda, J.-R. King, Y. Mi, M. Tsodyks, and S. Dehaene (2017). "A theory of working memory without consciousness or sustained activity." In: *Elife* 6, e23871 (cit. on p. 26).

Tsao, A., J. Sugar, L. Lu, C. Wang, J. J. Knierim, M. B. Moser, and E. I. Moser (2018). "Integrating time from experience in the lateral entorhinal cortex". In: *Nature* 561.7721, pp. 57–52 (cit. on p. 25).

Vasilaki, E., N. Frémaux, R. Urbanczik, W. Senn, and W. Gerstner (2009). "Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail". In: *PLoS computational biology* 5.12, e1000586 (cit. on pp. 51, 54).

Verstraeten, D., B. Schrauwen, M. D'Haene, and D. Stroobandt (2007). "An experimental unification of reservoir computing methods". In: *Neural Networks* 20.3. Echo State Networks and Liquid State Machines, pp. 391–403 (cit. on p. 30).

Vinyals, O., I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver (2019). *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/` (cit. on p. 56).

Volterra, V. (2005). *Theory of functionals and of integral and integro-differential equations*. Courier Corporation (cit. on p. 34).

Wang, J. X., Z. Kurth-Nelson, D. Kumaran, D. Tirumala, H. Soyer, J. Z. Leibo, D. Hassabis, and M. Botvinick (June 2018). "Prefrontal cortex as a meta-reinforcement learning system". en. In: *Nature Neuroscience* 21.6, pp. 860–868 (cit. on pp. 30, 31, 35, 42, 43, 51).

Wang, J. X., Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick (2016). "Learning to reinforcement learn". In: *arXiv preprint arXiv:1611.05763* (cit. on pp. 30, 31, 38, 43, 51).

Wang, Y., H. Markram, P. H. Goodman, T. K. Berger, J. Ma, and P. S. Goldman-Rakic (2006). "Heterogeneity in the pyramidal network of the medial prefrontal cortex". In: *Nature neuroscience* 9.4, p. 534 (cit. on pp. 15, 26, 69).

Warden, M. R. and E. K. Miller (2007). "The representation of multiple objects in prefrontal neuronal delay activity." In: *Cerebral Cortex* 17.suppl_1, pp. i41–i50 (cit. on p. 26).

Warden, P. (2018). "Speech commands: A dataset for limited-vocabulary speech recognition". In: *arXiv preprint arXiv:1804.03209* (cit. on p. 17).

Wasmuht, D. F., E. Spaak, T. J. Buschman, E. K. Miller, and M. G. Stokes (2018). "Intrinsic neuronal dynamics predict distinct functional roles during working memory". In: *Nature communications* 9.1, p. 3499 (cit. on pp. 14, 76).

Werbos, P. J. (Oct. 1990). "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10, pp. 1550–1560 (cit. on p. 1).

Winters, B. D., L. M. Saksida, and T. J. Bussey (2008). "Object recognition memory: neurobiological mechanisms of encoding, consolidation and retrieval." In: *Neuroscience & Biobehavioral Reviews* 32.5, pp. 1055–1070 (cit. on p. 28).

Wolff, M. J., J. Jochim, E. G. Akyürek, and M. G. Stokes (2017). "Dynamic hidden states underlying working-memory-guided behavior". In: *Nature Neuroscience* 20.6, p. 864 (cit. on pp. 14, 26, 27, 71).

Wolpert, D. M. and Z. Ghahramani (2000). "Computational principles of movement neuroscience". In: *Nature neuroscience* 3.11s, p. 1212 (cit. on p. 48).

Wong, J. D., D. A. Kistemaker, A. Chin, and P. L. Gribble (2012). "Can proprioceptive training improve motor learning?" In: *Journal of neurophysiology* 108.12, pp. 3313–3321 (cit. on p. 48).

Yagishita, S., A. Hayashi-Takagi, G. C. Ellis-Davies, H. Urakubo, S. Ishii, and H. Kasai (2014). "A critical time window for dopamine actions on the structural plasticity of dendritic spines". In: *Science* 345.6204, pp. 1616–1620 (cit. on p. 56).

Zador, A. M. (Aug. 2019). "A critique of pure learning and what artificial neural networks can learn from animal brains". en. In: *Nature Communications* 10.1. 00023, pp. 1–7 (cit. on pp. 3, 42, 43).