Gradient descent learning in recurrent and spiking neural network models

by

Guillaume Bellec

DISSERTATION

submitted for the degree of

Doctor Technicae

Institute of Theoretical Computer Science

Graz University of Technology

Thesis Advisor

Prof. Wolfgang Maass

Graz, June 2019

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

It remains unknown how complex animal behaviour emerges from the electric pulses exchanged between neurons in the brain. In contrast, very simplified mathematical models of neurons used in artificial intelligence (AI) are capable of playing chess and video games better than humans. For this to be possible the models of neural networks have diverged from what neurobiologists have been observing. In this thesis, we ask whether alternative neuron models that match more closely what is observed in the brain can use the learning theories developed in AI to produce intelligent behaviours. To do so, we first have to solve the technical and mathematical problems that were making learning difficult with neuron models which: (1) communicate solely via punctual electric pulses (spikes), (2) form loops of recurrent connections between each other, and (3) constantly rewire their connectivity structure to maintain a reduced number of active connections throughout learning. Quantitatively, our method reaches unpreceded computing capabilities with biological network models and approaches the performance standard AI models on benchmark tasks. Qualitatively, it can simulate animal-like behaviours requiring working memory and motor control. The second major contribution of this thesis is to suggest a new theory of learning in recurrent neural networks called eligibility propagation. When applied to brain models, it brings together how the connection between neurons are changing in experimental data and the learning theories of artificial network models. Altogether, we hope that our models will provide mathematical insights to inspire the progress of neuroscience in future years. More certainly, our algorithms are directly applicable to neuromorphic computers that substitute to usual computers by taking inspiration from the brain. We can already report today that our theories have contributed to the implementation of functional neural networks on such dedicated hardware and reduced the energy consumption of such models by two to three orders of magnitudes.

Zusammenfassung

Das aktuelle Verständnis unseres Gehirns kann noch nicht erklären, wie komplexe Verhaltensmuster aufgrund der elektrischen Impulse, die für die Kommunikation zwischen Nervenzellen im Gehirn verantwortlich sind, entstehen. Ganz im Gegensatz dazu können sehr simple mathematische Modelle von Nervenzellen, wie sie etwa im Gebiet der künstlichen Intelligenz (KI) verwendet werden, Schach und Videospiele besser spielen als ein Mensch. Um dies zu ermöglichen, musste man auf Modelle zurückgreifen, die den Beobachtungen von Neurobiologen eigentlich widersprechen. In dieser Doktorarbeit werden realistische Modelle von Nervenzellen verwendet, die besser mit den Beobachtungen im Gehirn übereinstimmen und mit Lerntheorien aus der KI ausgestattet, um intelligentes Verhalten hervorzubringen. Dies erfordert eine Lösung einer Reihe von technischen und mathematischen Problemen, weil Nervenzellen in solchen Modellen (1) nur vermöge punktierter elektrischer Impulse, "Spikes", miteinander kommunizieren, (2) Schleifen von rekurrenten Verbindungen ausbilden, und (3) kontinuierlich ihre Verbindungsstruktur neu verdrahten, um eine reduzierte Anzahl aktiver Verbindungen einzuhalten. Quantitativ wird aufgrund der hier entwickelten Methoden erstmals eine Lernfähigkeit erreicht, die bisher nur von abstrakten künstlichen Modellen erreicht wurde. Qualitativ können erstmals tierähnliche Fähigkeiten demonstriert werden, wie etwa Arbeitsgedächtnis oder motorische Fertigkeiten. Der zweite Beitrag dieser Doktorarbeit ist eine neue Theorie für das Lernen in rekurrenten Netzwerken von Nervenzellen: Die Ëignungspropagierung"(eligibility propagation). Für realistische Modelle von Nervenzellen im Gehirn kann erstmals eine Verbindung zwischen experimentellen Daten und Lernmethoden aus der KI hergestellt werden. Einerseits könnten die hier vorgestellten Modelle und Theorien den Fortschritt im Bereich der Neurowissenschaften in den kommenden Jahren vorantreiben und inspirieren, andererseits sind die hier vorgestellten Methoden direkt auf neuromorphen Computern anwendbar, die auf den Prinzipien der Informationsverarbeitung im Gehirn basieren. Bereits jetzt haben die hier vorgestellten Theorien zur Umsetzung von funktionalen neuronalen Netzen auf solcher Spezialhardware beigetragen, welche den Energieverbrauch von funktionellen Netzwerken um zwei bis drei Größenordnungen reduziert.

Résumé

Il n'est pas encore compris comment l'intelligence animale émerge à partir des courants électriques que s'échangent entre les neurones du cerveau. Pourtant en intelligence artificielle, des modèles mathématiques simplifiés de réseaux de neurones peuvent apprendre à jouer à des jeux vidéo ou conduire des voitures. Pour en arriver là, les modèles mathématiques se sont éloignés de la réalité observée par les neurobiologistes. Dans ce mémoire, nous nous demandons si des modèles de neurones plus réalistes peuvent être compatible avec les théories de l'apprentissage qui ont démontré leur succès en intelligence artificielle. Pour cela nous devons d'abord résoudre des problèmes techniques et mathématiques qui rendent l'apprentissage difficile pour des réseaux de neurones biologiques. C'est à dire, des neurones (1) qui communiquent de l'information uniquement via des courants électriques ponctuels (spikes), (2) dont les connexions forment des boucles récurrentes et (3) dont les connexions sont rares et se rebranchent continuellement. Quantitativement, notre nouvelle méthode atteint des capacités de calcul encore jamais observées pour des réseaux de neurones de ce type, ils deviennent pour la première fois compétitifs avec des modèles utilisés de façon canonique en intelligence artificielle. Qualitativement, nous arrivons à simuler des comportements qui ressemblent à celui de l'Animal et qui requièrent de la mémoire du travail et un contrôle moteur. Nous espérons que nos modèles mathématiques vont inspirer des progrès futurs en neurosciences. À plus court terme, nos modèles sont d'ores et déjà implémentés dans des prototypes d'ordinateur neuromorphique qui proposent une alternative aux ordinateurs traditionnels. Nous avons déjà observé que l'implementation de nos algorithmes sur des ordinateurs neuromorphiques permet de résoudre des problèmes d'intelligence artificielle en consommant 100 à 1000 fois moins d'énergie électrique qu'un ordinateur traditionnel.

Acknowledgements / Danksagung

I would like to thank my supervisor Wolfgang Maass for guiding me throughout my PhD with patience and flexibility. I am also grateful to Robert Legenstein who taught me with Wolfgang how to lead research with passion and mathematical rigour. Many thanks to my colleagues Anand Subramoney, David Kappel, Darjan Salaj, Franz Scherr, Arjun Rao, Chen Liu, and Elias Hajek who greatly contributed to the quality of my published work, and with whom I enjoyed a friendly teamwork experience. Many thanks to the administrative and technical staff Daniela Potzinger, Charlotte Rumpf, Oliver Friedl and Nicoletta Kähling who helped me on a daily basis. I would like to thank additionally Christopher Harvey, Christian Mayr, Mike Davies and Karlheinz Meier who gave me the opportunity to contribute to inspiring scientific collaborations. Many thanks to Jason MacLean, Jean Rouat, Yuqing Zhu, and Wulfram Gerstner for their valuable feedback on my work. Many thanks to my colleagues Franz Scherr, Thomas Limbacher, Darjan Salaj and Floran Unger for proof reading the present thesis. Special thanks to Alexander Just for helping with the translation of administrative and science related documents.

Merci a mon frère Pierre et mes parents Florence et Franęis pour m'avoir donné la force et les moyens d'en arriver là. Merci à toute la famille De Gennes pour m'avoir inspiré à démarrer une carrière de chercheur, et enfin merci aux professeurs de mathématiques Madame Rodoz et Messieurs Dubois, Perronet, Chateaux et Jean pour la qualité de leur enseignement et leur soutien.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

It is clear that the brain largely contributes to the "intelligence" of an animal. However, it remains a mystery how cognitive functions emerge from the neural circuits in the brain. Even simple functions like holding information in working memory for a few seconds are not well understood. Witnessing the recent success (LeCun et al., 2015) of simple mathematical models of network models ("artificial neural networks") which can learn to drive cars and play video games, it is tempting to believe that this mathematical framework can provide an understanding of the relationship between neurons and intelligence. Yet, there are many features of the brain that artificial neural networks do not model accurately. To model more closely the neurons in the brain we refine the realism of artificial network models and consider spiking neural networks that are sparsely and recurrently connected. Training such networks for solving a particular task was notoriously difficult. In this thesis, we have solve this technical challenge by extending the learning theories developed for artificial neural networks to this class of neural network models. It allows building networks that demonstrate cognitive capabilities on concrete tasks that were previously out of reach for such models. After defining what is meant by artificial models in Section 1.1 we summarize some of the experimental results supporting the refinement of our mathematical model in Sections 1.2 and 1.3. When then introduce in Section 1.4, an efficient training algorithm for these recurrent and spiking neural networks.

Beyond understanding how brain circuits compute, there is a gap between the learning theories in the artificial setting and the experimental data that accounts for learning mechanisms in the brain. This is partly because the experimental data about learning is very hard to collect as it requires to record brains activity across minutes, days, or years to capture physiological changes of the biological substrate while the learnt behaviour emerges. Despite this difficulty, some experimental paradigm has identified patterns in which connections between neurons strengthen, weaken or rewire. This phenomenon is known as synaptic plasticity and has been thought to underly learning capabilities. The other major contribution of this thesis introduced in Section 1.5 is to build an explicit link between learning theories from artificial intelligence and experimental data on synaptic plasticity.

1.1 Artificial neural network models

Most artificial neural network models can be defined as variants of the neural network as described below. The output z_j of an artificial neuron j combines a simple non-linear function σ with a linear combination of its inputs $\boldsymbol{x}$ weighted by the synaptic weights $\boldsymbol{w}_j$ such that $z_j = \sigma(\boldsymbol{w}_j \cdot \boldsymbol{x})$ (biases are omitted here for simplicity). In feedforward networks, artificial neurons are combined into consecutive layers, where the inputs onto each neuron are the neuron outputs of the preceding layer. The full computation of the deep neural network then defines one large mathematical function mainly parametrized by the weight matrices connecting each layer to the next.

One successful variant of a deep neural network is referred to as a convolutional network, there, one replaces the matrix-vector product by a product of convolution. This can be viewed as a replacement of the weight matrix by another one having much fewer parameters and exploiting invariant properties in the input structure. Even though successful in practice, it does not seem to hold a straightforward relationship with the connectivity structure observed in the brain. Hence we do not use a convolutional neural network to model biological neural networks in this thesis.

While a feedforward convolutional neural network is particularly useful to process static inputs without temporal dependencies such as images (LeCun et al., 2015), recurrent neural networks are often used to process time series. Unlike feedforward networks, the temporal dynamics of recurrent neural networks are modelled explicitly. Typically, the recurrent units send to each other their activity through a recurrent weight matrix to update the states of the same units at the next time step. A popular network model called "Long-short term memory" (LSTM) model (Hochreiter and Schmidhuber, 1997) also includes a hidden state in each network unit called a memory cell. Memory cells can hold information steadily over many consecutive time steps even in the absence of inputs. In the model, the commands for reading, forgetting or writing the memory cell content are formalized as sigmoidal gates that are part of the network model and trained simultaneously. Overall, recurrent neural networks are the natural choice for modelling neural circuits in this thesis for at least two reasons. Firstly, many interesting cognitive functions performed by the brain require temporal processing (e.g. evidence accumulation, decision making, language processing, speech recognition); secondly, it is an explicit model of the physical connections in the brain which are prominently recurrent (see Chapter 4 for a short review).

1.2 A simplified view of neural circuits in the brain

In contrast, a neuron in the brain consists of a cell body (soma), dendrites, and a single axon. The axon and dendrites are filaments that extrude from it. Dendrites typically branch profusely and extend a few hundred micrometres from the soma. The axon leaves the soma at a swelling and travels for as far as 1 meter in humans or more in other species. At the farthest tip of the axon, a signal can be transmitted to the dendrite of another cell (description adapted from the Wikipedia page about "neurons"). The connection between the axon and the dendrite of the subsequent neuron is called a synapse.

Spiking neuron models hold a central role in this thesis, but what is the electrophysiological reality that supports this modelling choice? A neuron at rest holds a differential of tens of millivolts between the inside and the outside of its membrane. In general, small depolarizations of this membrane potential do not propagate until the axon terminals and cannot communicate information to a subsequent neuron. Yet, if the voltage depolarization reaches a specific threshold level, voltage-gated ion channels open and provoke an action potential: a strong depolarization followed by a rapid hyperpolarization of the membrane potential (Hodgkin, 1958). This leads to a cascade of openings of neighbouring ion channels, and the action potential effectively travels all along the axon until it provokes the release of neurotransmitters at all the synapses formed with the subsequent neurons (Purves et al., 2008). This is often modeled as an all-or-none process and, as action potentials are rare (about 1 to 10Hz on average for human cortical neurons), neurons are thought to communicate in an event-based fashion. This simplified description summarizes what is thought as the most prominent mean of communication between neurons in many species including insects, humans and many more, even though other mechanisms have also been identified (Purves et al., 2008). To model spiking neurons in this thesis, we consider that the output of a spiking neuron model is binary: one when an action potential occurs and zero otherwise.

The brain is structured in different areas that have different functions and specificities. For instance, the connections across brain areas and within an area display stereotypical patterns shared across individuals of the same species. The role of this structure optimized through evolution is an important research question but we do not tackle this question in this present thesis. Here, we model biological neural networks as recurrently connected circuits formed by a few hundreds to a few thousands of neurons, with statistically homogenous wiring properties. In this very simplified network model, we attempt to understand some general properties of biological neural networks that remain poorly understood: recurrences, event-based computations and sparse connectivities. We typically think of such network model as a tiny portion of a brain area, but we remain unspecific and aim at a model capturing basic circuit properties that generalize across brain areas and across species.

Artificial neuron models are often connected in an all-to-all fashion, in contrast, neurons in the brain are more sparsely connected and many of their connections appear and disappear at the time scales of days (Trachtenberg et al., 2002). Moreover, brain circuits contain many neuron types with different morphologies, electrophysiological properties and connectivity patterns that are numerous even within a tiny volume of the cortex or the retina (Seung and Sümbül, 2014; Purves et al., 2008; Markram et al., 2015). We model this structure in a goal-driven optimization of the connectivity using the DEEP R algorithm described in Chapter 2. This rewiring algorithm optimizes the connectivity of sparsely connected neural networks to improve the network function. We show in particular that is can be used to model a network where neurons are split into two cell types: excitatory and inhibitory neurons (see Chapter 3). In this simplified model, spikes from excitatory and inhibitory neurons respectively depolarize or hyperpolarize the membrane of other neurons.

1.3 Modelling brain circuits with mathematical models of neural networks

By modelling the neural dynamics at different levels of details, different research questions can be tackled. At a macroscopic level, there are similarities between the representation of images in cortical areas and population of artificial neurons (Yamins and DiCarlo, 2016), but this type of artificial neuron model does not produce an accurate neuron-to-neuron mapping and it does not model the temporal processing of neuron at a precise temporal resolution. At another extreme, it is possible to model the activity of a single cell by describing the dynamics of ions flowing inside and outside a neuron (Hodgkin, 1958), and 3D model reconstruction of the cell morphology can capture the neural dynamics in more detail (Markram et al., 2015).

One intermediate approach is to aim at a simple mathematical neuron model which can still model accurately the neural dynamics with a resolution of a few milliseconds. The mathematical components of these models are rather simple: a spike generation process and linear temporal filters optimized for fitting the recorded data. Many popular spiking neuron models fall into this category, to list a few: the generalized linear model (GLM) (Pillow et al., 2008), the spike response model (SRM) (Gerstner et al., 2014) or variants of the leaky integrate and fire (LIF) neuron models (AllenInstitute, 2017). Due to the mathematical simplicity of these models, numerical optimization can be performed efficiently when fitting the model parameters to the data. This yields models which accurately predict the spike times of individual neurons (and sometimes the dynamics of the membrane voltage) without explicitly modelling their underlying molecular mechanisms nor the neuron morphology (Pillow et al., 2008; Brette and Gerstner, 2005; Pozzorini et al., 2015; Pozzorini et al., 2013; AllenInstitute, 2017).

In the present thesis, we choose a similarly simplified neuron model even though our goal is very different: we do not want to fit the model to experimental data but we want to optimize the model to serve a specific network function. Both problems require an optimization process that is prone to heavy computational loads and benefit from the simplicity of the neural model. The neuron model used in Chapters 3 and 4 typically integrates its inputs as a linear combination (like an artificial neuron), and it does not model the complexity of the dendritic tree. The temporal dynamics of the neuron model are then summarized with simple linear filters similarly as in the models listed above. We show in Chapter 3 that recurrent networks of such spiking neuron models can be trained and become competitive in comparison to the most popular artificial recurrent neural network models on machine learning benchmarks. It suggests that this neuron model is at the crossing between a model that can fit accurately recorded neural dynamics (Pillow et al., 2008; Pozzorini et al., 2015; Pozzorini et al., 2013; AllenInstitute, 2017), as well as a model that can support the optimization of the connection matrix to perform a function at the network level (see Chapter 3).

These models that we consider are variations of the LIF model with realistic time constants, but we did not aim at capturing the full complexity of the linear temporal filters that were previously used to predict experimental data. In our models, the slower dynamics of the membrane voltage is modelled with an exponential filter with decay time constants of about 20 ms of milliseconds (as captured with LIF models) and we include a model of adaptation to summarize the neural dynamics on the time scales of hundreds of milliseconds to a few seconds. Adaptation is a phenomenon characterized by a dampening of the firing rate of a neuron when stimulated with a constant current. This feature is widely spread in mammalian cortices, where about one-third of the neurons are substantially adaptive (Allen Institute: Cell Types Database, 2018). One simple model of adaptation is to consider that the firing threshold increases after each spike and slowly decays back to its resting value. This model of adaptation is indeed an essential component of the neuron model that can fit spike times accurately but adaptation can also be modelled in other ways (Pozzorini et al., 2015; AllenInstitute, 2017). When fitted to real neurons, the resulting adaptation time constants are of the order of hundreds of milliseconds which is much slower than the dynamics of the membrane potentials. Some data (Pozzorini et al., 2013; Pozzorini et al., 2015) even suggests that adaptation can hold significant effects after many seconds.

In Chapters 3 and 4 we use adaptation in our model to introduce slower dynamics and find that it supports working memory. Hence we refer to a recurrent network having a realistic proportion of adaptive neurons as a Long-short term memory Spiking Neural Network (LSNN), and this architecture serves as the backbone of the results achieved in Chapters 3 and 4. Other brain-inspired mechanisms like short-term dynamics of synapses (Tsodyks et al., 1998) have also been used to model short-term memory in previous models of recurrent networks of spiking neurons (Maass et al., 2002; Sussillo et al., 2007; Mongillo et al., 2008; Legenstein and Maass, 2007). Yet in these models, the recurrent connections are not optimized through learning. Neural dynamics like adaptation are cheaper in terms of computational load than simulating synaptic dynamics. This practical concern becomes particularly relevant when tackling learning problems which require to simulate the network for a large number of training trials.

1.4 Training recurrent networks of spiking neurons

To solve a pratical problem with any of the artificial neural network listed in Section 1.1 (e.g. image recognition, accumulating reward in a video game) one formalizes the learning problem by the minimization of a loss function E. For instance, it represents in supervised learning the mismatch between the output of the network for a given data sample and its target ouput. With artificial neural networks the error E is a differentiable function of the parameters of the neural networks and the parameters can be optimized by gradient descent. By adjusting the parameters by a small amount proportional to the gradient, the loss function reduces which reflects that the accuracy of the neural network increases. The optimization of the neural network is often referred to as "training" or "learning". In this thesis, we specificaly used "training" to refer to the optimization of neural networks with variants of gradient descent, whereas "learning" refers more generally to the capability of acquiring skills or knowledge. Gradient descent has shown to work well for training artificial network models, and backpropagation is the most common algorithm to compute the error gradients in artificial neural networks.

While artificial neural networks made a significant jump of performance in 2012 (Krizhevsky et al., 2012; LeCun et al., 2015), training spiking neurons with comparable efficiency have only been reported in the last couple of years (Esser et al., 2016; Bellec et al., 2019). This also reflects that there has been a technological barrier for building functional brain models that spike. One reason is that the typical formalism of spiking neural networks seems to be incompatible with gradient descent. Even though our choice of spiking network model remains simple, the notion of gradients in spiking neurons is problematic from a mathematical point of view. The generation of a spike is typically modelled as the binary event that happens when the membrane voltage crosses a threshold voltage. This threshold crossing condition results in a discountinuous mathematical model: a tiny change a network parameter might change the voltage history and generate or delete spikes and trigger cascades of events with large consequences. Due to these discountinuities, derivatives and gradients cannot be defined properly. Inspired by heuristics designed for artificial feedforward networks with binarized or discretized activations (Bengio et al., 2013; Raiko et al., 2014; Esser et al., 2016; Gu et al., 2015), we introduce a pseudo deriviative in Chapter 3 that allows the computation of gradients with back-propagtion through time (BPTT) in recurrent networks of spiking neurons. Remarkably this approximation works robustly in recurrent networks where gradient propagation is known to be much less stable (Bengio et al., 1994).

It seems that this training method improves upon the previous algorithms for training spiking neurons. In a recent publication (Bellec et al., 2019) we reproduced the hardest task solved with a recurrent spiking neural network and with FORCE learning (Sussillo and L. F. Abbott, 2009; Nicola and Clopath, 2017). We also reproduced the delayed XOR task solved with a contemporary formulation of BPTT for spiking neurons (Huh and Sejnowski, 2017). However, to compare more quantitatively these algorithms there is a need to report the performance on published benchmarks with well defined success measures. We report in Chapter 3 and 4 the performance of spiking neural networks on machine learning benchmarks, and we are not aware of another paper which reported a competitive performance with recurrent networks of spiking neurons on these problems. In comparison to artificial network models which are commonly benchmarked on temporal processing tasks such as speech recognition, it seems that LSNNs are the first recurrent spiking networks that achieve a competitive performance (see Chapters 3 and 4 and Bellec et al., 2019).

1.5 Modeling learning in the brain with gradient descent

Learning is a high-level cognitive function that relies on a combination of physiological mechanisms. One hypothesis is that many learning scenarios rely on the remapping on the connectivity in neural circuits or the strengthening of existing connections. This hypothesis was already formalulated by the neuroanatomist Cajal at the end of the 19th century. Later, Hebb conjectured in 1949 the existence of a simple rule describing how the connection strength changes, this is often summarized by the infamous sentence "cells that fire together wire together". Since then, series of experimental data have then shown that this conjecture happened to be suprisingly accurate, and the models of synaptic plasticity became more precise and quantitative.

One of the first experimental evidence of an activity dependent plasticity rule was found twenty years later (Bliss and Lomo, 1973). It was found that, when stimulating strongly a pre-synaptic neuron in the rabbit hippocampus, the transmission efficacy of the synapse increases and remains potentiated for hours. More recently, it was shown that when forcing repeatedly an action potential in the pre-synaptic neuron and in the post-synaptic neuron a few milliseconds later, one may induce a change of synaptic efficacy for which the precise spike timing matters (Gerstner et al., 1996; Bi and Poo, 1998). This experimental protocol alone leads in some cases to long-term changes of synaptic efficacy but, it can also be conditioned on the presence of neuromodulators such as dopamine (Schultz, 2002; Yagishita et al., 2014). This suggests that synaptic plasticity changes can be driven by pre-, post-synaptic and a third factor. Even if this third factor is most often thought to be a neuromodulator, other mechanisms such as plateau potentials can hold a similar function (see Frémaux and Gerstner, 2016 and Gerstner et al., 2018 for reviews).

Looking more closely on the relative timing of the third factor and the local activity (Yagishita et al., 2014), it was found that weight changes are still substantial when the dopamine arrives a few seconds after the activity induction. Along with other experiments reviewed in Gerstner et al., 2018, it suggests the presence of local mechanisms that retain traces of the recent activity during this temporal gap, there are often referred to as eligibility traces (Gerstner et al., 2018).

Simultaneously in computational neuroscience researchers have modelled how these mechanisms could support the emergence of complex learnt behaviours (Izhikevich, 2007; Legenstein et al., 2008; Kappel et al., 2018; Zenke and Ganguli, 2018; Frémaux et al., 2013). In consistence with the data suggesting the neurons releasing dopamine (dopaminergic neurons) are strongly correlated with reward prediction error (Schultz et al., 1997; Schultz, 2002; Engelhard et al., 2019). In models, the reward is used in models to formalize the task to be learnt and define the third factor. This was successful at solving goal-driven learning tasks of moderate difficulty. Yet, in comparison with machine learning algorithms relying on richer signals such as error back-propagation, it seems that data-inspired learning rules do not reach the performance of efficient learning algorithms used in artificial intelligence (see Lillicrap et al., 2016 and Chapter 4 for numerical quantitative comparisons). This seems to suggest that the current understanding of synaptic plasticity is not yet sufficient to explain the astounding learning capabilities of the brain, but Chapter 4 leads to a more optimistic conclusion.

Considering the recent success of gradient descent in artificial intelligence, some modellers have been looking for inspiration in artificial mathematical models. In artificial neural networks, the gradients are computed with error back-propagation, but it has been argued to be implausible that the brain implements this algorithm. Even in feedforward network models, back-propagation requires to propagate information backwards, mirroring the natural stream of information in the neural network, but such symmetric information pathways has not been observed in the brain. The search for a more plausible alternative to backpropagation in feedforward neural network is an active field of research, it was shown for instance that similar performance can be obtained when using: a fixed random network to propagate information backward (Lillicrap et al., 2016), or local plasticity rules with a forcing of the correct network output (Scellier and Bengio, 2017; Scellier and Bengio, 2019). Yet, none of these methods apply to recurrent neural networks. For recurrent network back-propagation is known as back-propagation through time (BPTT) and additionally requires to propagate information backwards in time. In Chapter 4 we suggest a plausible alternative to BPTT. On top of approaching the performance of BPTT on machine learning benchmarks, it results in weight updates that are compatible with the framework of three-factor learning rules modelling experimental data on synaptic plasticity.

The algorithm presented in Chapter 4 is called eligibility propagation (e-prop) because it relies on *eligibility traces* which capture the local traces of pre- and post-synpatic neural activities (Bellec et al., 2019). E-prop also considers a third factor called the *learning signal* which quantifies how the activity of the post-synaptic neuron influences the network error. This algorithm is proven in Chapter 4 to be mathematically equivalent to BPTT for a general class of recurrent neural networks including spiking (LSNNs) and artificial models (LSTM networks). For spiking models, the form of eligibility traces derived from the theory is multiplicative in the recent pre and postsynaptic activity which was also found to fit experimental data of synaptic plasticity accurately (Clopath et al., 2010). Empirically we show in simulations that it reduces substantially the performance gap between data-inspired learning algorithms and BPTT. Furthermore as a general theory of gradient descent, it applies to any type of learning tasks avoiding the design of learning rule hand crafted for a single learning task. In Chapter 4 and Bellec et al., 2019 we applied *e-prop* to supervised classification, regression, audio speech transcription and reward-based learning.

1.6 Perspectives for neuroscience

Solving the temporal credit assignment with predictive error signals and slow eligibility traces Using the e-prop theory, we derive in Chapter 4 eligibility traces for complex artificial and spiking model with enhanced working memory capabilities and apply them to tasks that require working memory. In particular for spiking neurons as illustrated in Chapter 4 and Bellec et al., 2019, the consideration of adaptation in the network model is decisive to solve evidence accumulation tasks where the sensory cues and the decisions are separated by a delay. Besides training efficient models with working memory, this models yields testable experimental predictions about the synaptic plasticity of adaptive neurons. Our theory predicts that the eligibility traces of adaptive neurons decay slower than for regular spiking neurons. A second prediction for adaptive neurons which is more quantitative is that the sign of the weight changes can flip between late and early arrivals of the learning signal.

The second class of problems where the temporal credit assignment is difficult are tasks where there is not only a delay between sensory cues and the decisions but also between actions reporting the decision and the rewards. This is modeled in a reinforcement learning task considered in (Bellec et al., 2019) where a sequence of motor actions leads potentially to a reward at the end of the trial. We found that e-prop can be combined with a policy gradient algorithm to solve this task (Bellec et al., 2019). There, the resulting learning signals combine reward prediction errors and neuron-specific feedbacks telling whether the recent actions are conservative or explorative.

The richness of learning signals These learning rules derived with e-prop are working well in simulatoins suggesting that third factors more complex than those considered in previous learning models can improve the learning performance significantly. First, the diversity or learning signals is important, second, even in the reinforcement learning context, better learning signals include information richer than a global reward prediction error. Going further, we illustrate in Chapter 4 that an external recurrent spiking neural network can emit structured learning signals optimized for learning a specific task rapidly.

From the side of experimental data, recent findings in neuroscience also support the existence of rich dopaminergic learning signals. It was found that the activity of VTA (ventral tegmental area) neurons encode not only the reward prediction error, but also other behaviourally relevant variables such as the subject's position or velocity inside a maze (Engelhard et al., 2019). Beyond these dopaminergic neurons, there seem to be more candidate mechanisms that can provide rich learning signals. There are for instance other neuromodulators and mechanisms that modulate synaptic plasticity (Gerstner et al., 2018), and performance monitoring neurons are also found prominently in the cortices (Sajad et al., 2019). Altogether, these neuroscience experiments and our mathematical models converge to the idea that further investigations on the learning signals are likely to advance the understanding of neuroscience and artificial intelligence.

1.7 Persectives for machine learning and neuromorphic hardware

Recurrent neural networks Even in standard recurrent neural networks BPTT costs a large amount of memory. BPTT is particularly problematic when processing long time-series. There, if the entire time horizon does not fit in memory or if it is better to implement intermediate weight updates, BPTT is replaced by truncated-BPTT. This algorithm suffers in particular from the truncation of the temporal context used to compute the gradients. The theory of e-prop suggests an alternative for computing gradients in recurrent neural networks, and it seems to be competitive with other alternatives to BPTT (for an exhaustive comparison, we refer to Bellec et al., 2019). We believe that the most promising extension of e-prop for machine learning as it combines the best features of truncated-BPTT with the eligibility traces defined by e-prop. This algorithm is described in Chapter 4 under the name of e-prop 3, and it substitutes to truncated-BPTT for training recurrent neural networks. We demonstrate there that it is particularly efficient when the temporal context considered in BPTT does not contain sufficient information about the network history. As of now, Chapter 4 provides a proof of concept but more simulations are required to demonstrate the competitivity of this algorithm on hard machine learning problems.

Energy efficient computing with neuromorphic hardware The brain-inspired techniques developed through out this thesis provide a different paradigm for implementing neural networks. Currently, neural networks are often simulated on GPUs and CPUs with low-level software interfaces optimized for the multiplication of large dense matrices. It means in particular that the current hardware hardly benefits from the introduction of structured connectivities through sparse matrices in the network architecture. On the other hand, brain-inspired hardware with a dedicated implementation of sparse matrices can benefit from a sparse and structured connectivity. In fact, we provided a proof of concept of this statement in Liu et al., 2018 by implementing the algorithm DEEP R from Chapter 2 for training deep sparse feedforward neural networks on a prototype of the SpiNNaker 2 neuromorphic chip. This implementation of hardware and software, led to a neural network for hand-written digit recognition that is trained with two order of magnitude fewer energy consumption in comparison with a conventional CPU.

Besides the usage of sparse connectivities, the nature of spiking neurons is also beneficial for energy efficiency on dedicated hardware. To demonstrate this we ported a trained recurrent neural network of spiking neurons onto the Loihi neuromorphic chip developed by Intel (Davies et al., 2018). When training the network with precautions, the performance of the neural network is not degraded when porting it from a digital computer to the neuromorphic chip. This allows to pre-train the network model off-line, and use the trained model with an energy-efficient neuromorphic chip. A publication summarizing these results is currently in preparation.

A neuromorphic hardware that spikes and learns via e-prop The rapid development of neuromorphic computers leave the hope that the spiking network technology will reach another level of computational performance when switching from conventional computers to dedicated hardware. It is already clear that spiking hardware consumes much less energy, this might make it possible to scale up the network size and reach computational compatibilities in-approachable with artificial neural network due to the high energetical cost of conventional hardware.

A singular feature of the recent Loihi chip (Davies et al., 2018) is to enable on-chip learning in a way that is inspired by the brain. Following the tradition of three-factor learning rules, it is already possible to implement reward gated synaptic plasticity rule on this chip (Davies et al., 2018). This means that most of the requirement for implementing e-prop in dedicated hardware is already available. It promises that upcoming neuromorphic chips will be able to implement e-prop efficiently, which offers an algorithm that competes with BPTT without requiring its demanding memory management. Altogether, it makes neuromorphic hardware a technology that might open new horizons that cannot be forseen with to the current learning paradigm used in artificial intelligence.

Chapter 2

Deep Rewiring: Training very sparse deep networks

Contents

Neuromorphic hardware tends to pose limits on the connectivity of deep networks that one can run on them. But also generic hardware and software implementations of deep learning run more efficiently for sparse networks. Several methods exist for pruning connections of a neural network after it was trained without connectivity constraints. We present an algorithm, DEEP R, that enables us to train directly a sparsely connected neural network. DEEP R automatically rewires the network during supervised training so that connections are there where they are most needed for the task, while its total number is all the time strictly bounded. We demonstrate that DEEP R can be used to train very sparse feedforward and recurrent neural networks on standard benchmark tasks with just a minor loss in performance. DEEP R is based on a rigorous theoretical foundation that views rewiring as stochastic sampling of network configurations from a posterior.

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

Guillaume Bellec, David Kappel, Wolfgang Maass, Robert Legenstein (2018). "Deep Rewiring: Training very sparse deep networks." *International Conference on Learning Representations.*

To this study, I contributed as first author. The study was conceived by GB, DK, WM and RL, with the theory being developed by GB and DK. The experiments were designed by GB, DK, WM and RL, and were conducted by GB. The core of the manuscript (sections 1 to 5) was mainly written by RL but all authors contributed to the writing.

2.1 Introduction

Network connectivity is one of the main determinants for whether a neural network can be efficiently implemented in hardware or simulated in software. For example, it is mentioned in Jouppi et al., 2017 that in Google's tensor processing units (TPUs), weights do not normally fit in on-chip memory for neural network applications despite the small 8 bit weight precision on TPUs. Memory is also the bottleneck in terms of energy consumption in TPUs and FPGAs (Han et al., 2017; Iandola et al., 2016). For example, for an implementation of a long short term memory network (LSTM), memory reference consumes more than two orders of magnitude more energy than ALU operations (Han et al., 2017). The situation is even more critical in neuromorphic hardware, where either hard upper bounds on network connectivity are unavoidable (Schemmel et al., 2010; Merolla et al., 2014) or fast on-chip memory of local processing cores is severely limited, for example the 96 MByte local memory of cores in the SpiNNaker system (Furber et al., 2014). This implementation bottleneck will become even more severe in future applications of deep learning when the number of neurons in layers will increase, causing a quadratic growth in the number of connections between them.

Evolution has apparently faced a similar problem when evolving large neuronal systems such as the human brain, given that the brain volume is dominated by white matter, i.e., by connections between neurons. The solution found by evolution is convincing. Synaptic connectivity in the brain is highly dynamic in the sense that new synapses are constantly rewired, especially during learning (Holtmaat et al., 2005; Stettler et al., 2006; Attardo et al., 2015; Chambers and Rumpel, 2017). In other words, rewiring is an integral part of the learning algorithms in the brain, rather than a separate process.

We are not aware of previous methods for simultaneous training and rewiring in artificial neural networks, so that they are able to stay within a strict bound on the total number of connections throughout the learning process. There are however several heuristic methods for pruning a larger network (Han et al., 2015b; Han et al., 2015a; Collins and Kohli, 2014; Z. Yang et al., 2015; Srinivas and Babu, 2015), that is, the network is first trained to convergence, and network connections and / or neurons are pruned only subsequently. These methods are useful for downloading a trained network on neuromorphic hardware, but not for on-chip training. A number of methods have been proposed that are capable of reducing connectivity during training (Collins and Kohli, 2014; Jin et al., 2016; Narang et al., 2017). However, these algorithms usually start out with full connectivity. Hence, besides reducing computational demands only partially, they cannot be applied when computational resources (such as memory) is bounded throughout training.

Inspired by experimental findings on rewiring in the brain, we propose in this article deep rewiring (DEEP R), an algorithm that makes it possible to train deep neural networks under strict connectivity constraints. In contrast to many previous pruning approaches that were based on heuristic arguments, DEEP R is embedded in a thorough theoretical framework. DEEP R is conceptually different from standard gradient descent algorithms in two respects. First, each connection has a predefined sign. Specifically, we assign to each connection k a connection parameter θ_k and a constant sign $s_k \in \{-1, 1\}$. For non-negative θ_k, the corresponding network weight is given by $w_k = s_k \theta_k$. In standard backprop, when the absolute value of a weight is moved through 0, it becomes a weight with the opposite sign. In contrast, in DEEP R a connection vanishes in this case ($w_k = 0$), and a randomly drawn other connection is tried out by the algorithm. Second, in DEEP R, gradient descent is combined with a random walk in parameter space (Freitas et al., 2000; Welling and Teh, 2011). This modification leads to important functional differences. In fact, our theoretical analysis shows that DEEP R jointly samples network weights and the network architecture (i.e., network connectivity) from the posterior distribution, that is, the distribution that combines the data likelihood and a specific connectivity prior in a Bayes optimal manner. As a result, the algorithm continues to rewire connections even when the performance has converged. We show that this feature enables DEEP R to adapt the network connectivity structure online when the task demands are drifting.

We show on several benchmark tasks that with DEEP R, the connectivity of several deep architectures — fully connected deep networks, convolutional nets, and recurrent networks (LSTMs) — can be constrained to be extremely sparse throughout training with a marginal drop in performance. In one example, a standard feed forward network trained on the MNIST dataset, we achieved good performance with 2 % of the connectivity of the fully connected counterpart. We show that DEEP R reaches a similar performance level as state-of-the-art pruning algorithms where training starts with the full connectivity matrix. If the target connectivity is very sparse (a few percent of the full connectivity), DEEP R outperformed these pruning algorithms.

2.2 Rewiring in deep neural networks

Stochastic gradient descent (SGD) and its modern variants (Kingma and Ba, 2014; Tieleman and Hinton, 2012) implemented through the Error Backpropagation algorithm is the dominant learning paradigm of contemporary deep learning applications. For a given list of network inputs $\mathbf{X}$ and target network outputs $\mathbf{Y}^*$, gradient descent iteratively moves the parameter vector $\boldsymbol{\theta}$ in the direction of the negative gradient of an error function $E_{\mathbf{X},\mathbf{Y}^*}(\boldsymbol{\theta})$ such that a local minimum of $E_{\mathbf{X},\mathbf{Y}^*}(\boldsymbol{\theta})$ is eventually reached.

A more general view on neural network training is provided by a probabilistic interpretation of the learning problem (Bishop, 2006; Neal, 1992). In this probabilistic learning framework, the deterministic network output is interpreted as defining a probability distribution $p_{\mathcal{N}}(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta})$ over outputs $\mathbf{Y}$ for the given input

$\mathbf{X}$ and the given network parameters $\boldsymbol{\theta}$. The goal of training is then to find parameters that maximize the likelihood $p_{\mathcal{N}}\left(\mathbf{Y}^* \,|\, \mathbf{X}, \boldsymbol{\theta}\right)$ of the training targets under this model (maximum likelihood learning). Training can again be performed by gradient descent on an equivalent error function that is usually given by the negative log-likelihood $E_{\mathbf{X},\mathbf{Y}^*}(\boldsymbol{\theta}) = -\log p_{\mathcal{N}}\left(\mathbf{Y}^* \,|\, \mathbf{X}, \boldsymbol{\theta}\right)$.

Going one step further in this reasoning, a full Bayesian treatment adds prior beliefs about the network parameters through a prior distribution $p_{\mathcal{S}}\left(\boldsymbol{\theta}\right)$ (we term this distribution the structural prior for reasons that will become clear below) over parameter values $\boldsymbol{\theta}$ and the training goal is formulated via the posterior distribution over parameters $\boldsymbol{\theta}$. The training goal that we consider in this article is to produce sample parameter vectors which have a high probability under the posterior distribution $p^*(\boldsymbol{\theta} \,|\, \mathbf{X}, \mathbf{Y}^*) \propto p_{\mathcal{S}}\left(\boldsymbol{\theta}\right) \cdot p_{\mathcal{N}}\left(\mathbf{Y}^* \,|\, \mathbf{X}, \boldsymbol{\theta}\right)$. More generally, we are interested in a target distribution $p^*(\boldsymbol{\theta}) \propto p^*(\boldsymbol{\theta} \,|\, \mathbf{X}, \mathbf{Y}^*)^{\frac{1}{T}}$ that is a tempered version of the posterior where T is a temperature parameter. For $T = 1$ we recover the posterior distribution, for $T > 1$ the peaks of the posterior are flattened, and for $T < 1$ the distribution is sharpened, leading to higher probabilities for parameter settings with better performance.

This training goal was explored by Welling and Teh, 2011, C. Chen et al., 2016, and Kappel et al., 2015 where it was shown that gradient descent in combination with stochastic weight updates performs Markov Chain Monte Carlo (MCMC) sampling from the posterior distribution. In this paper we extend these results by (a) allowing the algorithm also to sample the network structure, and (b) including a hard posterior constraint on the total number of connections during the sampling process. We define the training goal as follows:

$$\text{produce samples } \boldsymbol{\theta} \text{ with high probability in } \; p^*(\boldsymbol{\theta}) \;=\; \begin{cases} 0 \text{ if } \boldsymbol{\theta} \text{ violates the constraint} \\ \frac{1}{\mathcal{Z}} p^*(\boldsymbol{\theta} \,|\, \mathbf{X}, \mathbf{Y}^*)^{\frac{1}{T}} \text{ otherwise,} \end{cases} \tag{2.1}$$

where $\mathcal{Z}$ is a normalizing constant. The emerging learning dynamics jointly samples from a posterior distribution over network parameters $\boldsymbol{\theta}$ and constrained network architectures. In the next section we introduce the algorithm and in Section 2.4 we discuss the theoretical guarantees.

The DEEP R algorithm: In many situations, network connectivity is strictly limited during training, for instance because of hardware memory limitations. Then the limiting factor for a training algorithm is the maximal connectivity ever needed during training. DEEP R guarantees such a hard limit. DEEP R achieves the learning goal (2.1) on *network configurations*, that is, it not only samples the network weights and biases, but also the connectivity under the given constraints. This is achieved by introducing the following mapping from network parameters $\boldsymbol{\theta}$ to network weights $\mathbf{w}$:

A connection parameter θ_k and a constant sign $s_k \in \{-1, 1\}$ are assigned to each connection k. If θ_k is negative, we say that the connection k is *dormant*, and the corresponding weight is $w_k = 0$. Otherwise, the connection is considered *active*, and the corresponding weight is $w_k = s_k \theta_k$. Hence, each $\boldsymbol{\theta}_k$ encodes (a) whether the connection is active in the network, and (b) the weight of the connection if it is active. Note that we use here a single index k for each connection / weight instead of the more usual double index that defines the sending and receiving neuron. This connection-centric indexing is more natural for our rewiring algorithms where the connections are in the focus rather than the neurons. Using this mapping, sampling from the posterior over $\boldsymbol{\theta}$ is equivalent to sampling from the posterior over network configurations, that is, the network connectivity structure and the network weights.

```
1  for $i$ in $[1, N_{iterations}]$ do
2      for all active connections $k$ $(\theta_k \geq 0)$ do
3          $\theta_k \leftarrow \theta_k - \eta \frac{\partial}{\partial \theta_k} E_{\mathbf{X},\mathbf{Y}^*}(\boldsymbol{\theta}) - \eta\alpha + \sqrt{2\eta T}\ \nu_k$;
4          if $\theta_k < 0$ then set connection $k$ dormant ;
5      end
6      while number of active connections lower than $K$ do
7          select a dormant connection $k'$ with uniform probability and activate
           it;
8          $\theta_{k'} \leftarrow 0$
9      end
10 end
```

Algorithm 1: Pseudo code of the DEEP R algorithm. ν_k is sampled from a zero-mean Gaussian of unit variance independently for each active and each update step. Note that the gradient of the error $E_{\mathbf{X},\mathbf{Y}^*}(\boldsymbol{\theta})$ is computed by backpropagation over a mini-batch in practice.

DEEP R is defined in Algorithm 1. Gradient updates are performed only on parameters of active connections (line 3). The derivatives of the error function $\frac{\partial}{\partial \theta_k} E_{\mathbf{X},\mathbf{Y}^*}(\boldsymbol{\theta})$ can be computed in the usual way, most commonly with the backpropagation algorithm. Since we consider only classification problems in this article, we used the cross-entropy error for the experiments in this article. The third term in line 3 $(-\eta\alpha)$ is an ℓ_1 regularization term, but other regularizers could be used as well.

A conceptual difference to gradient descent is introduced via the last term in line 3. Here, noise $\sqrt{2\eta T}\ \nu_k$ is added to the update, where the temperature parameter T controls the amount of noise and ν_k is sampled from a zero-mean Gaussian of unit variance independently for each parameter and each update step. The last term alone would implement a random walk in parameter space. Hence, the whole line 3 of the algorithm implements a combination of gradient descent on the regularized error function with a random walk. Our theoretical analysis shows that this random walk behavior has an important functional consequence, see the paragraph after the next for a discussion on the theoretical properties of DEEP R.

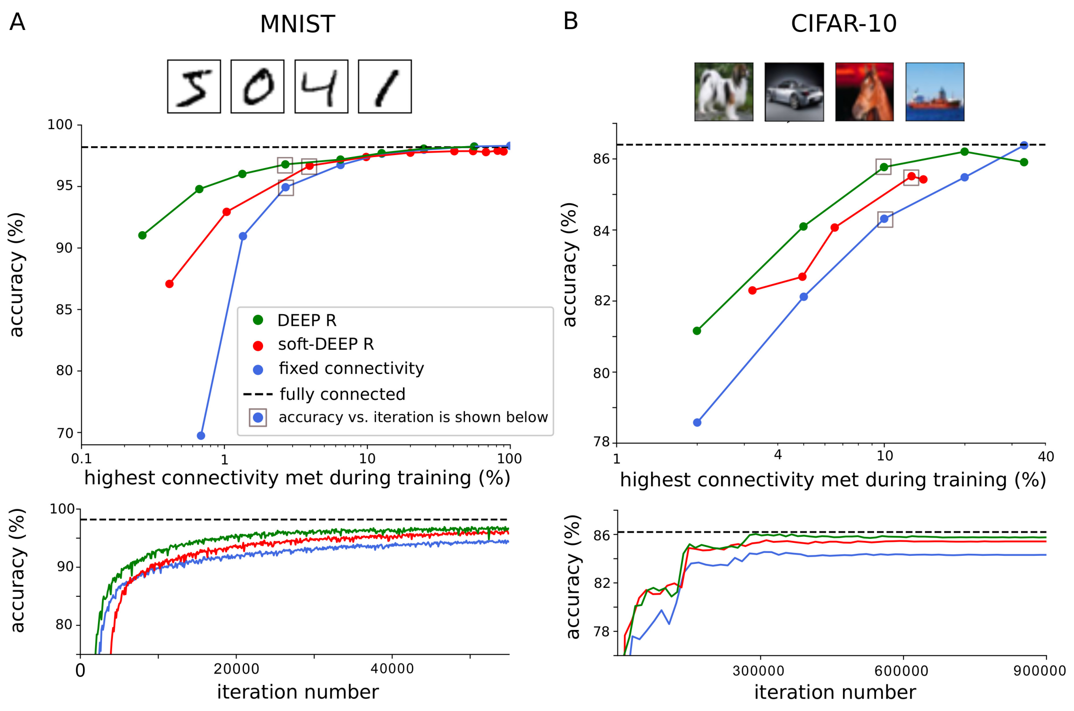


Fig. 2.1: Visual pattern recognition with sparse networks during training. Sample training images (top), test classification accuracy after training for various connectivity levels (middle) and example test accuracy evolution during training (bottom) for a standard feed forward network trained on MNIST (**A**) and a CNN trained on CIFAR-10 (**B**). Accuracies are shown for various algorithms. Green: DEEP R; red: soft-DEEP R; blue: SGD with initially fixed sparse connectivity; dashed gray: SGD, fully connected. Since soft-DEEP R does not guarantee a strict upper bound on the connectivity, accuracies are plotted against the highest connectivity ever met during training (middle panels). Iteration number refers to the number of parameter updates during training.

The rewiring aspect of the algorithm is captured in lines 4 and 6–9 in Algorithm (1). Whenever a parameter θ_k becomes smaller than 0, the connection is set dormant, i.e., it is deleted from the network and no longer considered for updates (line 4). For each connection that was set to the dormant state, a new connection k' is chosen randomly from the uniform distribution over dormant connections, k' is activated and its parameter is initialized to 0. This rewiring strategy (a) ensures that exactly K connections are active at any time during training (one initializes the network with K active connections), and (b) that dormant connections do not need any computational demands except for drawing connections to be activated. Note that for sparse networks, it is efficient to keep only a list of active connections and none for the dormant connections. Then, one can efficiently draw connections from the whole set of possible connections and reject those that are already active.

2.3 Experiments

Rewiring in fully connected and in convolutional networks: We first tested the performance of DEEP R on MNIST and CIFAR-10. For MNIST, we considered a fully connected feed-forward network used in Han et al., 2015b to benchmark pruning algorithms. It has two hidden layers of 300 and 100 neurons respectively and a 10-fold softmax output layer. On the CIFAR-10 dataset, we used a convolutional neural network (CNN) with two convolutional followed by two fully connected layers. For reproducibility purposes the network architecture and all parameters of this CNN were taken from the official tutorial of Tensorflow. On CIFAR-10, we used a decreasing learning rate and a cooling schedule to reduce the temperature parameter T over iterations (see Appendix B.1 for details on all experiments).

For each task, we performed four training sessions. First, we trained a network with DEEP R. In the CNN, the first convolutional layer was kept fully connected while we allowed rewiring of the second convolutional layer. Second, we tested another algorithm, soft-DEEP R, which is a simplified version of DEEP R that does however not guarantee a strict connectivity constraint (see Section 2.4 for a description). Third, we trained a network in the standard manner without any rewiring or pruning to obtain a baseline performance. Finally, we trained a network with a connectivity that was randomly chosen before training and kept fixed during the optimization. The connectivity was however not completely random. Rather each layer received a number of connections that was the same as the number found by soft-DEEP R. The performance of this network is expected to be much better than a network where all layers are treated equally.

Fig. 2.1 shows the performance of these algorithms on MNIST (panel A) and on CIFAR-10 (panel B). DEEP R reaches a classification accuracy of 96.2 % when constrained to 1.3 % connectivity. To evaluate precisely the accuracy that is reachable with 1.0 % connectivity, we did an additional experiment where we doubled the number of training epochs. DEEP R reached a classification accuracy of 96.3% (less than 2 % drop in comparison to the fully connected baseline). Training on fixed random connectivity performed surprisingly well for connectivities around 10 %, possibly due to the large redundancy in the MNIST images. Soft-DEEP R does not guarantee a strict upper bound on the network connectivity. When considering the maximum connectivity ever seen during training, soft-DEEP R performed consistently worse than DEEP R for networks where this maximum connectivity was low. On CIFAR-10, the classification accuracy of DEEP R was 84.1 % at a connectivity level of 5 %. The performance of DEEP R at 20 % connectivity was close to the performance of the fully connected network.

To study the rewiring properties of DEEP R, we monitored the number of newly activated connections per iteration (i.e., connections that changed their status from dormant to active in that iteration). We found that after an initial transient, the number of newly activated connections converged to a stable value and remained stable even after network performance has converged, see Appendix B.2.

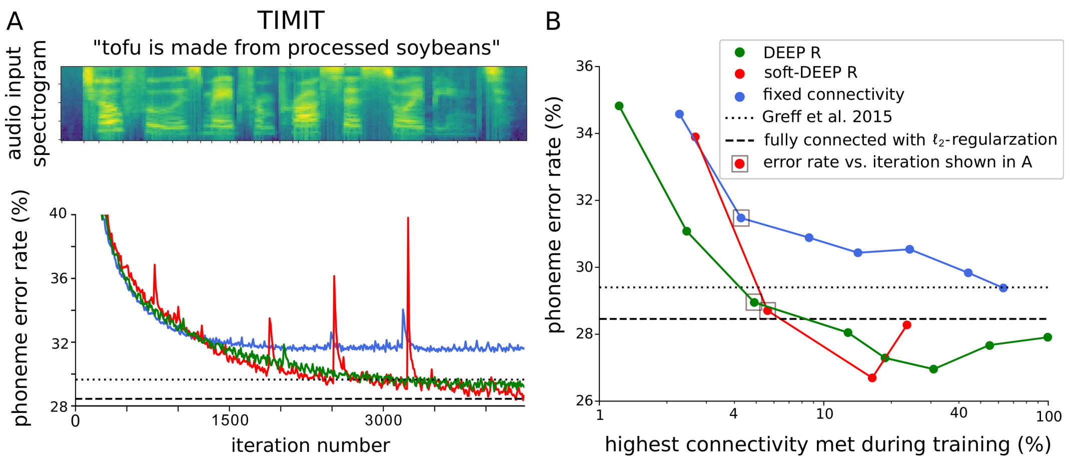


Fig. 2.2: Rewiring in recurrent neural networks. Network performance for one example run (**A**) and at various connectivity levels (**B**) as in Fig. 2.1 for an LSTM network trained on the TIMIT dataset with DEEP R (green), soft-DEEP R (red) and a network with fixed random connectivity (blue). Dotted line: fully connected LSTM trained without regularization as reported in Greff et al., 2017. Thick dotted line: fully connected LSTM with ℓ_2 regularization.

Rewiring in recurrent neural networks: In order to test the generality of our rewiring approach, we also considered the training of recurrent neural networks with backpropagation through time (BPTT). Recurrent networks are quite different from their feed forward counterparts in terms of their dynamics. In particular, they are potentially unstable due to recurrent loops in inference and training signals. As a test bed, we considered an LSTM network trained on the TIMIT data set. In our rewiring algorithms, all connections were potentially available for rewiring, including connections to gating units. From the TIMIT audio data, MFCC coefficients and their temporal derivatives were computed and fed into a bi-directional LSTMs with a single recurrent layer of 200 cells followed by a softmax to generate the phoneme likelihood (Graves and Schmidhuber, 2005), see Appendix B.1.

We considered as first baseline a fully connected LSTM with standard BPTT without regularization as the training algorithm. This algorithm performed similarly as the one described in Greff et al., 2017. It turned out however that performance could be significantly improved by including a regularizer in the training objective. We therefore considered the same setup with ℓ_2 regularization (cross-validated). This setup achieved a phoneme error rate of 28.3 %. We note that better results have been reported in the literature using the CTC cost function and deeper networks (Graves et al., 2013). For the sake of easy comparison however, we sticked here to the much simpler setup with a medium-sized network and the standard cross-entropy error function.

We found that connectivity can be reduced significantly in this setup with our algorithms, see Fig. 2.2. Both algorithms, DEEP R and soft-DEEP R, performed even slightly better than the fully connected baseline at connectivities around 10 %, probably due to generalization issues. DEEP R outperformed soft-DEEP R at

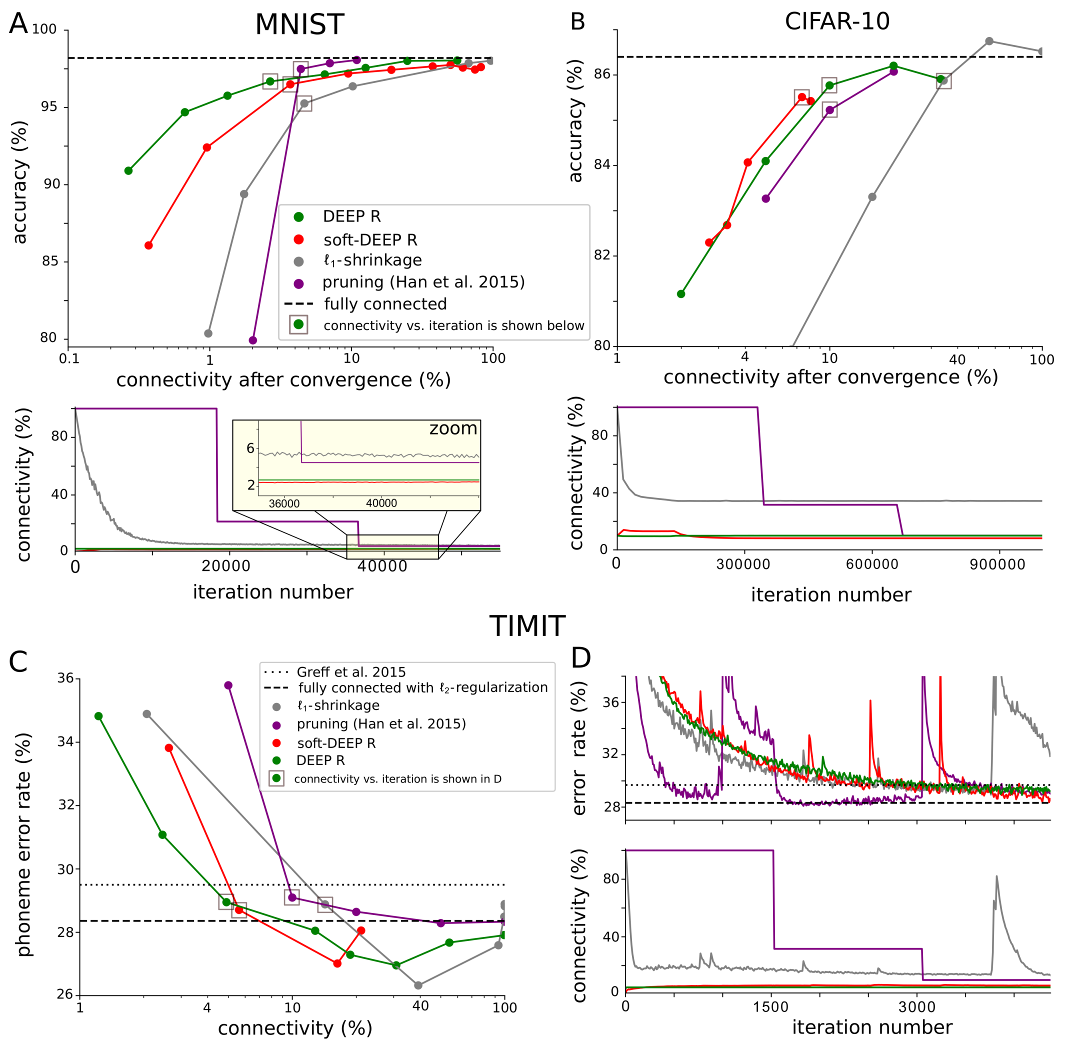


Fig. 2.3: Efficient network solutions under strict sparsity constraints. Accuracy and connectivity obtained by DEEP R and soft-DEEP R in comparison to those achieved by pruning (Han et al., 2015b) and ℓ_1-shrinkage (Tibshirani, 1996; Collins and Kohli, 2014). **A, B)** Accuracy against the connectivity for MNIST (A) and CIFAR-10 (B). For each algorithm, one network with a decent compromise between accuracy and sparsity is chosen (small gray boxes) and its connectivity across training iterations is shown below. **C)** Performance on the TIMIT dataset. **D)** Phoneme error rates and connectivities across iteration number for representative training sessions.

very low connectivities and it outperformed BPTT with fixed random connectivity consistently at any connectivity level considered.

Comparison to algorithms that cannot be run on very sparse networks: We wondered how much performance is lost when a strict connectivity constraint has to be taken into account during training as compared to pruning algorithms

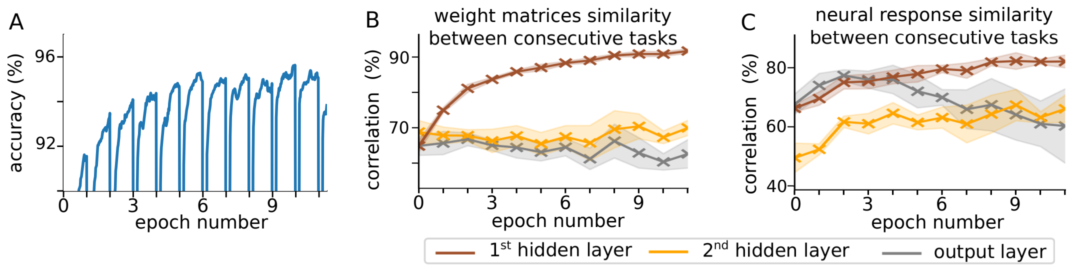


Fig. 2.4: Transfer learning with DEEP R. The target labels of the MNIST data set were shuffled after every epoch. **A)** Network accuracy vs. training epoch. The increase of network performance across tasks (epochs) indicates a transfer of knowledge between tasks. **B)** Correlation between weight matrices of subsequent epochs for each network layer. **C)** Correlation between neural activity vectors of subsequent epochs for each network layer. The transfer is most visible in the first hidden layer, since weights and outputs of this layer are correlated across tasks. Shaded areas in **B)** and **C)** represent standard deviation across 5 random seeds, influencing network initialization, noisy parameter updates, and shuffling of the outputs.

that only achieve sparse networks after training. To this end, we compared the performance of DEEP R and soft-DEEP R to recently proposed pruning algorithms: ℓ_1-shrinkage (Tibshirani, 1996; Collins and Kohli, 2014) and the pruning algorithm proposed by Han et al., 2015b. ℓ_1-shrinkage uses simple ℓ_1-norm regularization and finds network solutions with a connectivity that is comparable to the state of the art (Collins and Kohli, 2014; Yu et al., 2012). We chose this one since it is relatively close to DEEP R with the difference that it does not implement rewiring. The pruning algorithm from Han et al., 2015b is more complex and uses a projection of network weights on a ℓ_0 constraint. Both algorithms prune connections starting from the fully connected network. The hyper-parameters such as learning rate, layer size, and weight decay coefficients were kept the same in all experiments. We validated by an extensive parameter search that these settings were good settings for the comparison algorithms, see Appendix B.1.

Results for the same setups as considered above (MNIST, CIFAR-10, TIMIT) are shown in Fig. 2.3. Despite the strict connectivity constraints, DEEP R and soft-DEEP R performed slightly better than the unconstrained pruning algorithms on CIFAR-10 and TIMIT at all connectivity levels considered. On MNIST, pruning was slightly better for larger connectivities. On MNIST and TIMIT, pruning and ℓ_1-shrinkage failed completely for very low connectivities while rewiring with DEEP R or soft-DEEP R still produced reasonable networks in this case.

One interesting observation can be made for the error rate evolution of the LSTM on TIMIT (Fig. 2.3D). Here, both ℓ_1-shrinkage and pruning induced large sudden increases of the error rate, possibly due to instabilities induced by parameter changes in the recurrent network. In contrast, we observed only small glitches of this type in DEEP R. This indicates that sparsification of network connectivity is harder in recurrent networks due to potential instabilities, and that DEEP R is better suited to avoid such instabilities. The reason for this advantage of DEEP R is however not clear.

Transfer learning is supported by DEEP R: If the temperature parameter T is kept constant during training, the proposed rewiring algorithms do not converge to a static solution but explore continuously the posterior distribution of network configurations. As a consequence, rewiring is expected to adapt to changes in the task in an on line manner. If the task demands change in an online learning setup, one may hope that a transfer of invariant aspects of the tasks occurs such that these aspects can be utilized for faster convergence on later tasks (transfer learning). To verify this hypothesis, we performed one experiment on the MNIST dataset where the class to which each output neuron should respond to was changed after each training epoch (class-shuffled MNIST task). Fig. 2.4A shows the performance of a network trained with DEEP R in the class-shuffled MNIST task. One can observe that performance recovered after each shuffling of the target classes. More importantly, we found a clear trend of increasing classification accuracy even across shuffles. This indicates a form of transfer learning in the network such that information about the previous tasks (i.e., the previous target-shuffled MNIST instances) was preserved in the network and utilized in the following instances. We hypothesized for the reason of this transfer that early layers developed features that were invariant to the target shuffling and did not need to be re-learned in later task instances. To verify this hypothesis, we computed the following two quantities. First, in order to quantify the speed of parameter dynamics in different layers, we computed the correlation between the layer weight matrices of two subsequent training epoch (Fig. 2.4B). Second, in order to quantify the speed of change of network dynamics in different layers, we computed the correlation between the neuron outputs of a layer in subsequent epochs (Fig. 2.4C). We found that the correlation between weights and layer outputs increased across training epochs and were significantly larger in early layers. This supports the hypothesis that early network layers learned features invariant to the shuffled coding convention of the output layer.

2.4 Convergence properties of DEEP R and soft-DEEP R

The theoretical analysis of DEEP R is somewhat involved due to the implemented hard constraints. We therefore first introduce and discuss here another algorithm, soft-DEEP R where the theoretical treatment of convergence is more straight forward. In contrast to standard gradient-based algorithms, this convergence is not a convergence to a particular parameter vector, but a convergence to the target distribution over network configurations.

Convergence properties of soft-DEEP R: The soft-DEEP R algorithm is given in Algorithm 2. Note that the updates for active connections are the same as for DEEP R (line 3). Also the mapping from parameters θ_k to weights w_k is the same as in DEEP R. The main conceptual difference to DEEP R is that connection parameters continue their random walk when dormant (line 7). Due to this random walk,

```
1  for $i$ in $[1, N_{iterations}]$ do
2      for all active connections $k$ $(\theta_k \geq 0)$ do
3          $\theta_k \leftarrow \theta_k - \eta \frac{\partial}{\partial \theta_k} E_{\mathbf{X},\mathbf{Y}^*}(\boldsymbol{\theta}) - \eta\alpha + \sqrt{2\eta T}\ \nu_k$;
4          if $\theta_k < 0$ then set connection $k$ dormant ;
5      end
6      for all dormant connections $k$ $(\theta_k < 0)$ do
7          $\theta_k \leftarrow \theta_k + \sqrt{2\eta T}\ \nu_k$;
8          $\theta_k \leftarrow \max\{\theta_k, \theta_{\min}\}$;
9          if $\theta_k \geq 0$ then set connection $k$ active ;
10     end
11 end
```

Algorithm 2: Pseudo code of the soft-DEEP R algorithm. $\theta_{\min} < 0$ is a constant that defines a lower boundary for negative θ_ks.

connections will be re-activated at random times when they cross zero. Therefore, soft-DEEP R does not impose a hard constraint on network connectivity but rather uses the ℓ_1 norm regularization to impose a soft-constraint.

Since dormant connections have to be simulated, this algorithm is computationally inefficient for sparse networks. An approximation could be used where silent connections are re-activated at a constant rate, leading to an algorithm very similar to DEEP R. DEEP R adds to that the additional feature of a strict connectivity constraint.

The central result for soft-DEEP R has been proven in the context of spiking neural networks in (Kappel et al., 2015) in order to understand rewiring in the brain from a functional perspective. The same theory however also applies to standard deep neural networks. To be able to apply standard mathematical tools, we consider parameter dynamics in continuous time. In particular, consider the following stochastic differential equation (SDE)

$$d\theta_k \;=\; \beta\ \frac{\partial}{\partial \theta_k} \log p^*(\boldsymbol{\theta}|\mathbf{X},\mathbf{Y}^*)\Big|_{\boldsymbol{\theta}^t} dt \;+\; \sqrt{2\beta T}\, d\mathcal{W}_k\ , \tag{2.2}$$

where β is the equivalent to the learning rate and $\frac{\partial}{\partial \theta_k} \log p^*(\boldsymbol{\theta}|\mathbf{X},\mathbf{Y}^*)\Big|_{\boldsymbol{\theta}^t}$ denotes the gradient of the log parameter posterior evaluated at the parameter vector $\boldsymbol{\theta}^t$ at time t. The term $d\mathcal{W}_k$ denotes the infinitesimal updates of a standard Wiener process. This SDE describes gradient ascent on the log posterior combined with a random walk in parameter space. We show in Appendix B.3 that the unique stationary distribution of this parameter dynamics is given by

$$p^*(\boldsymbol{\theta}) = \frac{1}{\mathcal{Z}} p^*(\boldsymbol{\theta}\,|\,\mathbf{X},\mathbf{Y}^*)^{\frac{1}{T}}\ . \tag{2.3}$$

Since we considered classification tasks in this article, we interpret the network output as a multinomial distribution over class labels. Then, the derivative of the log likelihood is equivalent to the derivative of the negative cross-entropy error. Together with an ℓ_1 regularization term for the prior, and after discretization of time, we obtain the update of line 3 in Algorithm 2 for non-negative parameters. For negative parameters, the first term in Eq. (2.2) vanishes since the network weight is constant zero there. This leads to the update in line 7. Note that we introduced a reflecting boundary at $\theta_{\min} < 0$ in the practical algorithm to avoid divergence of parameters (line 8).

Convergence properties of DEEP R: A detailed analysis of the stochastic process that underlies the algorithm is provided in Appendix B.4. Here we summarize the main findings. Each iteration of DEEP R in Algorithm 1 consists of two parts: In the first part (lines 2-5) all connections that are currently active are advanced, while keeping the other parameters at 0. In the second part (lines 6-9) the connections that became dormant during the first step are randomly replenished.

To describe the connectivity constraint over connections we introduce the binary constraint vector $\mathbf{c}$ which represents the set of active connections, i.e., element c_k of $\mathbf{c}$ is 1 if connection k is allowed to be active and zero else. In Theorem 2 of Appendix B.4, we link DEEP R to a compound Markov chain operator that simultaneously updates the parameters $\boldsymbol{\theta}$ according to the soft-DEEP R dynamics under the constraint $\mathbf{c}$ and the constraint vector $\mathbf{c}$ itself. The stationary distribution of this Markov chain is given by the joint probability

$$p^*(\boldsymbol{\theta}, \mathbf{c}) \;\propto\; p^*(\boldsymbol{\theta})\, \mathcal{C}(\boldsymbol{\theta}, \mathbf{c})\, p_{\mathcal{C}}(\mathbf{c}) \,, \tag{2.4}$$

where $\mathcal{C}(\boldsymbol{\theta}, \mathbf{c})$ is a binary function that indicates compatibility of $\boldsymbol{\theta}$ with the constraint $\mathbf{c}$ and $p^*(\boldsymbol{\theta})$ is the tempered posterior of Eq. (2.3) which is left stationary by soft-DEEP R in the absence of constraints. $p_{\mathcal{C}}(\mathbf{c})$ in Eq. (2.4) is a uniform prior over all connectivity constraints with exactly K synapses that are allowed to be active. By marginalizing over $\mathbf{c}$, we obtain that the posterior distribution of DEEP R is identical to that of soft-DEEP R if the constraint on the connectivity is fulfilled. By marginalizing over $\boldsymbol{\theta}$, we obtain that the probability of sampling a network architecture (i.e. a connectivity constraint $\mathbf{c}$) with DEEP R and soft-DEEP R are proportional to one another. The only difference is that DEEP R exclusively visits architectures with K active connections (see equation (B.35) in Appendix B.4 for details).

In other words, DEEP R solves a constraint optimization problem by sampling parameter vectors $\boldsymbol{\theta}$ with high performance within the space of constrained connectivities. The algorithm will therefore spend most time in network configurations where the connectivity supports the desired network function, such that, connections with large support under the objective function (2.1) will be maintained active with high probability, while other connections are randomly tested and discarded if found not useful.

2.5 Discussion

Related Work: Freitas et al., 2000 considered sequential Monte Carlo sampling to train neural networks by combining stochastic weight updates with gradient updates. Stochastic gradient updates in mini-batch learning was considered in Welling and Teh, 2011, where also a link to the true posterior distribution was established. C. Chen et al., 2016 proposed a momentum scheme and temperature annealing (for the temperature T in our notation) for stochastic gradient updates, leading to a stochastic optimization method. DEEP R extends this approach by using stochastic gradient Monte Carlo sampling not only for parameter updates but also to sample the connectivity of the network. In addition, the posterior in DEEP R is subject to a hard constraint on the network architecture. In this sense, DEEP R performs constrained sampling, or constrained stochastic optimization if the temperature is annealed. S. Patterson and Teh, 2013 considered the problem of stochastic gradient dynamics constrained to the probability simplex. The methods considered there are however not readily applicable to the problem of constraints on the connection matrix considered here. Additionally, we show that a correct sampler can be constructed that does not simulate dormant connections. This sampler is efficient for sparse connection matrices. Thus, we developed a novel method, random reintroduction of connections, and analyzed its convergence properties (see Theorem 2 in Appendix B.4).

Conclusions: We have presented a method for modifying backprop and backprop-through-time so that not only the weights of connections, but also the connectivity graph is simultaneously optimized during training. This can be achieved while staying always within a given bound on the total number of connections. When the absolute value of a weight is moved by backprop through 0, it becomes a weight with the opposite sign. In contrast, in DEEP R a connection vanishes in this case (more precisely: becomes dormant), and a randomly drawn other connection is tried out by the algorithm. This setup requires that, like in neurobiology, the sign of a weight does not change during learning. Another essential ingredient of DEEP R is that it superimposes the gradient-driven dynamics of each weight with a random walk. This feature can be viewed as another inspiration from neurobiology (Mongillo et al., 2017). An important property of DEEP R is that — in spite of its stochastic ingredient — its overall learning dynamics remains theoretically tractable: Not as gradient descent in the usual sense, but as convergence to a stationary distribution of network configurations which assigns the largest probabilities to the best-performing network configurations. An automatic benefit of this ongoing stochastic parameter dynamics is that the training process immediately adjusts to changes in the task, while simultaneously transferring previously gained competences of the network (see Fig. 2.4).

Acknowledgements Written under partial support by the Human Brain Project of the European Union #720270, and the Austrian Science Fund (FWF): I 3251-N33.

We thank Franz Pernkopf and Matthias Zöhrer for useful comments regarding the TIMIT experiment.

Chapter 3

Long short-term memory and learning-to-learn in networks of spiking neurons

Contents

Recurrent networks of spiking neurons (RSNNs) underlie the astounding computing and learning capabilities of the brain. But computing and learning capabilities of RSNN models have remained poor, at least in comparison with artificial neural networks (ANNs). We address two possible reasons for that. One is that RSNNs in the brain are not randomly connected or designed according to simple rules, and they do not start learning as a tabula rasa network. Rather, RSNNs in the brain were optimized for their tasks through evolution, development, and prior experience. Details of these optimization processes are largely unknown. But their functional contribution can be approximated through powerful optimization methods, such as backpropagation through time (BPTT).

A second major mismatch between RSNNs in the brain and models is that the latter only show a small fraction of the dynamics of neurons and synapses in the brain. We include neurons in our RSNN model that reproduce one prominent dynamical process of biological neurons that takes place at the behaviourally relevant time scale of seconds: neuronal adaptation. We denote these networks as LSNNs because of their Long short-term memory. The inclusion of adapting neurons drastically increases the computing and learning capability of RSNNs if they are trained and configured by deep learning (BPTT combined with a rewiring algorithm that optimizes the network architecture). In fact, the computational performance of these RSNNs approaches for the first time that of LSTM networks. In addition RSNNs with adapting neurons can acquire abstract knowledge from prior learning in a Learning-to-Learn (L2L) scheme, and transfer that knowledge in order to learn new but related tasks from very few examples. We demonstrate this for supervised learning and reinforcement learning.

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

> 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.*

To this study, GB, DS and AS contributed as first authors. All authors contributed to the conception of the study and the design of the experiments. The implementation of back-propagation through time for spiking recurrent networks was implemented by GB, and the code was used in all other experiments. The sequential MNIST and TIMIT experiments were implemented by GB and DS. The supervised LTL experiments were implemented by AS. The Meta-RL experiment was implemented by GB. The core of the manuscript (sections 1 to 7) was mainly written by WM but all authors contributed to the writing.

3.1 Introduction

Recurrent networks of spiking neurons (RSNNs) are frequently studied as models for networks of neurons in the brain. In principle, they should be especially well-suited for computations in the temporal domain, such as speech processing, as their computations are carried out via spikes, i.e., events in time and space. But the performance of RSNN models has remained suboptimal also for temporal processing tasks. One difference between RSNNs in the brain and RSNN models is that RSNNs in the brain have been optimized for their function through long evolutionary processes, complemented by a sophisticated learning curriculum during development. Since most details of these biological processes are currently still unknown, we asked whether deep learning is able to mimic these complex optimization processes on a functional level for RSNN models. We used BPTT as the deep learning method for network optimization. Backpropagation has been adapted previously for feed forward networks with binary activations in (Courbariaux et al., 2016; Esser et al., 2016), and we adapted BPTT to work in a similar manner for RSNNs. In order to also optimize the connectivity of RSNNs, we augmented BPTT with DEEP R, a biologically inspired heuristic for synaptic rewiring (Kappel et al., 2018; Bellec et al., 2018a). Compared to LSTM networks, RSNNs tend to have inferior short-term memory capabilities. Since neurons in the brain are equipped with a host of dynamics processes on time scales larger than a few dozen ms (Hasson et al., 2015), we enriched the inherent dynamics of neurons in our model by a standard neural adaptation process.

We first show (section 3.4) that this approach produces new computational performance levels of RSNNs for two common benchmark tasks: Sequential MNIST and TIMIT (a speech processing task). We then show that it makes L2L applicable to RSNNs (section 3.5), similarly as for LSTM networks. In particular, we show that meta-RL (J. X. Wang et al., 2016; Duan et al., 2016) produces new motor control capabilities of RSNNs (section 3.6). This result links a recent abstract model for reward-based learning in the brain J. X. Wang et al., 2018 to spiking activity. In addition, we show that RSNNs with sparse connectivity and sparse firing activity of 10-20 Hz (see Fig. 1D, 2D, S1C) can solve these and other tasks. Hence these RSNNs compute with spikes, rather than firing rates.

The superior computing and learning capabilities of LSNNs suggest that they are also of interest for implementation in spike-based neuromorphic chips such as Brainscales (Schemmel et al., 2010), SpiNNaker (Furber et al., 2013), True North (Esser et al., 2016), chips from ETH Zürich (Qiao et al., 2015), and Loihi (Davies et al., 2018). In particular, nonlocal learning rules such as backprop are challenges for some of these neuromorphic devices (and for many brain models). Hence alternative methods for RSNN learning of nonlinear functions are needed. We show in sections 3.5 and 3.6 that L2L can be used to generate RSNNs that learn very efficiently even in the absence of synaptic plasticity.

Relation to prior work: We refer to (Eliasmith, 2013; DePasquale et al., 2016; Huh and Sejnowski, 2017; Nicola and Clopath, 2017) for summaries of preceding results on computational capabilities of RSNNs. The focus there was typically on the generation of dynamic patterns. Such tasks are not addressed in this article, but it will be shown in (Bellec et al., 2018b) that LSNNs provide an alternative model to Nicola and Clopath, 2017 for the generation of complex temporal patterns. Huh et al. (Huh and Sejnowski, 2017) applied gradient descent to recurrent networks of spiking neurons. There, neurons without a leak were used. Hence, the voltage of a neuron could used in that approach to store information over an unlimited length of time.

We are not aware of previous attempts to bring the performance of RSNNs for time series classification into the performance range of LSTM networks. We are also not aware of any previous literature on applications of L2L to SNNs.

3.2 LSNN model

Neurons and synapses in common RSNN models are missing many of the dynamic processes found in their biological counterparts, especially those on larger time scales. We integrate one of them into our RSNN model: neuronal adaptation. It is well known that a substantial fraction of excitatory neurons in the brain are adapting, with diverse time constants, see e.g. the Allen Brain Atlas for data from the neocortex of mouse and humans. We refer to the resulting type of RSNNs as Long short-term memory Spiking Neural Networks (LSNNs). LSNNs consist of a population R of integrate-and-fire (LIF) neurons (excitatory and inhibitory), and a second population A of LIF excitatory neurons whose excitability is temporarily reduced through preceding firing activity, i.e., these neurons are adapting (see Fig. 3.1C and Suppl.). Both populations R and A receive spike trains from a population X of external input neurons. Results of computations are read out by a population Y of external linear readout neurons, see Fig. 3.1C.

Common ways for fitting models for adapting neurons to data are described in (Gerstner et al., 2014; Pozzorini et al., 2015; N. W. Gouwens et al., 2018; Teeter et al., 2018). We are using here the arguably simplest model: We assume that the firing threshold $B_j(t)$ of neuron j increases by some fixed amount $\beta/\tau_{a,j}$ for each spike of this neuron j, and then decays exponentially back to a baseline value b_j^0 with a time constant $\tau_{a,j}$. Thus the threshold dynamics for a discrete time step of $\delta t = 1$ ms reads as follows

$$\begin{aligned} B_j(t) &= b_j^0 + \beta b_j(t), & (3.1)\\ b_j(t+\delta t) &= \rho_j b_j(t) + (1-\rho_j) z_j(t), & (3.2)\end{aligned}$$

where $\rho_j = \exp(-\frac{\delta t}{\tau_{a,j}})$ and $z_j(t)$ is the spike train of neuron j assuming values in $\{0, \frac{1}{\delta t}\}$. Note that this dynamics of thresholds of adaptive spiking neurons is similar to the dynamics of the state of context neurons in (Mikolov et al., 2014). It generally suffices to place the time constant of adapting neurons into the desired range for short-term memory (see Suppl. for specific values used in each experiment).

3.3 Applying BPTT with DEEP R to RSNNs and LSNNs

We optimize the synaptic weights, and in some cases also the connectivity matrix of an LSNN for specific ranges of tasks. The optimization algorithm that we use, backpropagation through time (BPTT), is not claimed to be biologically realistic. But like evolutionary and developmental processes, BPTT can optimize LSNNs for specific task ranges. Backpropagation (BP) had already been applied in (Courbariaux et al., 2016) and (Esser et al., 2016) to feedforward networks of spiking neurons. In these approaches, the gradient is backpropagated through spikes by replacing the non-existent derivative of the membrane potential at the time of a spike by a pseudo-derivative that smoothly increases from 0 to 1, and then decays back to 0. We reduced (“dampened”) the amplitude of the pseudo-derivative by a factor < 1 (see Suppl. for details). This enhances the performance of BPTT for RSNNs that compute during larger time spans, that require backpropagation through several 1000 layers of an unrolled feedforward network of spiking neurons. A similar implementation of BPTT for RSNNs was proposed in (Huh and Sejnowski, 2017). It is not yet clear which of these two versions of BPTT work best for a given task and a given network.

In order to optimize not only the synaptic weights of a RSNN but also its connectivity matrix, we integrated BPTT with the biologically inspired (Kappel et al.,

2018) rewiring method DEEP R (Bellec et al., 2018a) (see Suppl. for details). DEEP R converges theoretically to an optimal network configuration by continuously updating the set of active connections (Kappel et al., 2015; Kappel et al., 2018; Bellec et al., 2018a).

3.4 Computational performance of LSNNs

Sequential MNIST: We tested the performance of LSNNs on a standard benchmark task that requires continuous updates of short term memory over a long time span: sequential MNIST (Le et al., 2015; Costa et al., 2017). We compare the performance of LSNNs with that of LSTM networks. The size of the LSNN, in the case of full connectivity, was chosen to match the number of parameters of the LSTM network. This led to 120 regular spiking and 100 adaptive neurons (with adaptation time constant τ_a of 700 ms) in comparison to 128 LSTM units. Actually it turned out that the sparsely connected LSNN shown in Fig. 3.1C, which was generated by including DEEP R in BPTT, had only 12% of the synaptic connections but performed better than the fully connected LSNN (see "DEEP R LSNN" versus "LSNN" in Fig. 3.1B).

The task is to classify the handwritten digits of the MNIST dataset when the pixels of each handwritten digit are presented sequentially, one after the other in 784 steps, see Fig. 3.1A. After each presentation of a handwritten digit, the network is required to output the corresponding class. The grey values of pixels were given directly to artificial neural networks (ANNs), and encoded by spikes for RSNNs. We considered both the case of step size 1 ms (requiring 784 ms for presenting the input image) and 2 ms (requiring 1568 ms for each image, the adaptation time constant τ_a was set to 1400 ms in this case, see Fig. 3.1B.). The top row of Fig. 3.1D shows a version where the grey value of the currently presented pixel is encoded by population coding through the firing probability of the 80 input neurons. Somewhat better performance was achieved when each of the 80 input neurons is associated with a particular threshold for the grey value, and this input neuron fires whenever the grey value crosses its threshold in the transition from the previous to the current pixel (this input convention is chosen for the SNN results of Fig. 3.1B). In either case, an additional input neuron becomes active when the presentation of the 784 pixel values is finished, in order to prompt an output from the network. The firing of this additional input neuron is shown at the top right of the top panel of Fig. 3.1D. The softmax of 10 linear output neurons Y is trained through BPTT to produce, during this time segment, the label of the sequentially presented handwritten digit. We refer to the yellow shading around 800 ms of the output neuron for label 3 in the plot of the dynamics of the output neurons Y in Fig. 3.1D. This output was correct.

A performance comparison is given in Fig. 3.1B. LSNNs achieve 94.7% and 96.4% classification accuracy on the test set when every pixel is presented for 1 and 2ms respectively. An LSTM network achieves 98.5% and 98.0% accuracy on the same task

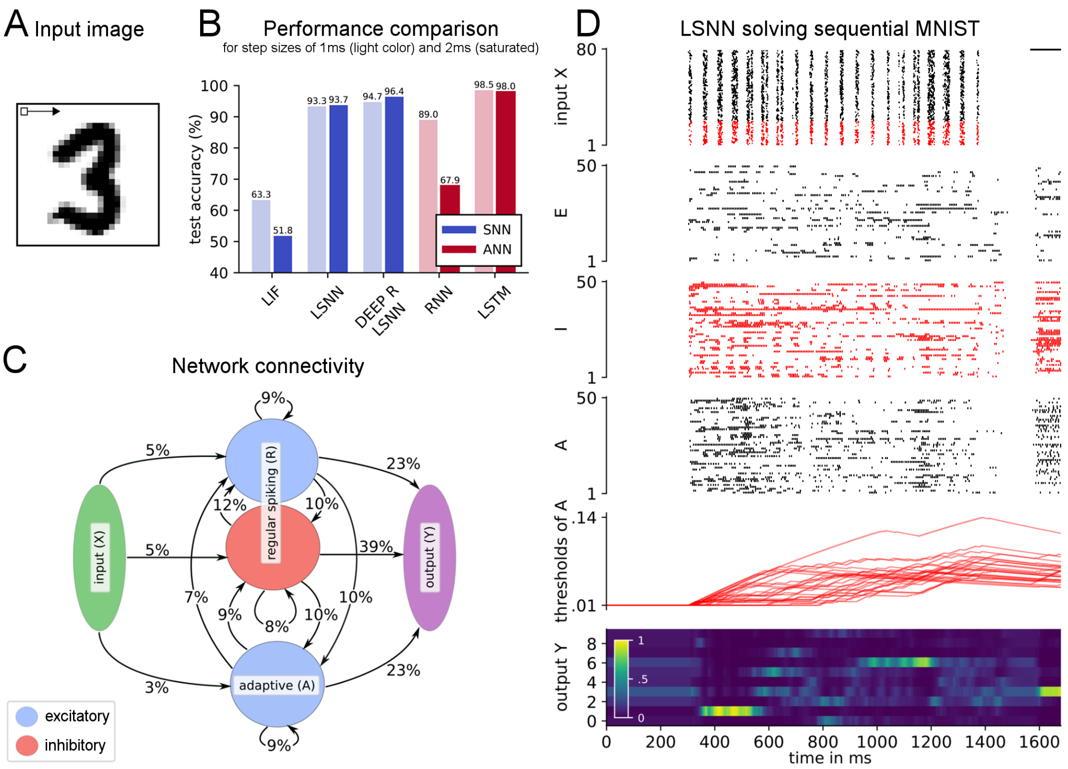


Fig. 3.1: Sequential MNIST. A The task is to classify images of handwritten digits when the pixels are shown sequentially pixel by pixel, in a fixed order row by row. **B** The performance of RSNNs is tested for three different setups: without adapting neurons (LIF), a fully connected LSNN, and an LSNN with randomly initialized connectivity that was rewired during training (DEEP R LSNN). For comparison, the performance of two ANNs, a fully connected RNN and an LSTM network are also shown. **C** Connectivity (in terms of connection probabilities between and within the 3 subpopulations) of the LSNN after applying DEEP R in conjunction with BPTT. The input population X consisted of 60 excitatory and 20 inhibitory neurons. Percentages on the arrows from X indicate the average connection probabilities from excitatory and inhibitory neurons. **D** Dynamics of the LSNN after training when the input image from A was sequentially presented. From top to bottom: spike rasters from input neurons (X), and random subsets of excitatory (E) and inhibitory (I) regularly spiking neurons, and adaptive neurons (A), dynamics of the firing thresholds of a random sample of adaptive neurons; activation of softmax readout neurons.

setups. The LIF and RNN bars in Fig. 3.1B show that this accuracy is out of reach for BPTT applied to spiking or nonspiking neural networks without enhanced short term memory capabilities. We observe that in the sparse architecture discovered by DEEP R, the connectivity onto the readout neurons Y is denser than in the rest of the network (see Fig. 3.1C). Detailed results are given in the supplement.

Speech recognition (TIMIT): We also tested the performance of LSNNs for a real-world speech recognition task, the TIMIT dataset. A thorough study of the performance of many variations of LSTM networks on TIMIT has recently been carried out in (Greff et al., 2017). We used exactly the same setup which was used there (framewise classification) in order to facilitate comparison. We found that a standard LSNN consisting of 300 regularly firing (200 excitatory and 100 inhibitory) and 100 excitatory adapting neurons with an adaptation time constant of 200 ms, and with 20% connection probability in the network, achieved a classification error of 33.2%. This error is below the mean error around 40% from 200 trials with different hyperparameters for the best performing (and most complex) version of LSTMs according to Fig. 3 of (Greff et al., 2017), but above the mean of 29.7% of the 20 best performing choices of hyperparameters for these LSTMs. The performance of the LSNN was however somewhat better than the error rates achieved in (Greff et al., 2017) for a less complex version of LSTMs without forget gates (mean of the best 20 trials: 34.2%).

We could not perform a similarly rigorous search over LSNN architectures and meta-parameters as was carried out in (Greff et al., 2017) for LSTMs. But if all adapting neurons are replaced by regularly firing excitatory neurons one gets a substantially higher error rate than the LSNN with adapting neurons: 37%. Details are given in the supplement.

3.5 LSNNs learn-to-learn from a teacher

One likely reason why learning capabilities of RSNN models have remained rather poor is that one usually requires a tabula rasa RSNN model to learn. In contrast, RSNNs in the brain have been optimized through a host of preceding processes, from evolution to prior learning of related tasks, for their learning performance. We emulate a similar training paradigm for RSNNs using the L2L setup. We explore here only the application of L2L to LSNNs, but L2L can also be applied to RSNNs without adapting neurons (Subramoney et al., 2018). An application of L2L to LSNNs is tempting, since L2L is most commonly applied in machine learning to their ANN counterparts: LSTM networks see e.g. (J. X. Wang et al., 2016; Duan et al., 2016). LSTM networks are especially suited for L2L since they can accommodate two levels of learning and representation of learned insight: Synaptic connections and weights can encode, on a higher level, a learning algorithm and prior knowledge on a large time-scale. The short-term memory of an LSTM network can accumulate, on a lower level of learning, knowledge during the current learning task. It has recently been argued J. X. Wang et al., 2018 that the pre-frontal cortex

(PFC) similarly accumulates knowledge during fast reward-based learning in its short-term memory, without using dopamine-gated synaptic plasticity, see the text to Suppl. Fig. 3 in (J. X. Wang et al., 2018). The experimental results of Perich et al., 2018 suggest also a prominent role of short-term memory for fast learning in the motor cortex.

The standard setup of L2L involves a large, in fact in general infinitely large, family $\mathcal{F}$ of learning tasks C. Learning is carried out simultaneously in two loops (see Fig. 3.2A). The *inner loop* learning involves the learning of a single task C by a neural network $\mathcal{N}$, in our case by an LSNN. Some parameters of $\mathcal{N}$ (termed hyper-parameters) are optimized in an *outer loop* optimization to support fast learning of a randomly drawn task C from $\mathcal{F}$. The outer loop training – implemented here through BPTT – proceeds on a much larger time scale than the inner loop, integrating performance evaluations from many different tasks C of the family $\mathcal{F}$. One can interpret this outer loop 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, but the term training is less descriptive of the longer-term evolutionary processes we mimic. Like in (Hochreiter et al., 2001; J. X. Wang et al., 2016; Duan et al., 2016) we let all synaptic weights of $\mathcal{N}$ belong to the set of hyper-parameters that are optimized through 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 and the thresholds of adapting neurons. Thus the synaptic weights of the neural network $\mathcal{N}$ are free to encode an efficient *algorithm* for learning arbitrary tasks C from $\mathcal{F}$.

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. The L2L results of (Hochreiter et al., 2001) show that LSTM networks can learn nonlinear functions from a teacher without modifying their synaptic weights, using their short-term memory instead. We asked whether this form of learning can also be attained by LSNNs.

Task: We considered the task of learning complex non-linear functions from a teacher. Specifically, we chose as family $\mathcal{F}$ of tasks a class of continuous functions of two real-valued variables (x_1, x_2). 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 from [-1, 1], see Fig. 3.2B. 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. We gave the teacher input to the LSNN for learning a particular TN C from $\mathcal{F}$ in a delayed manner as in (Hochreiter et al., 2001): The target output value was given after $\mathcal{N}$ had provided its guessed output value for the preceding input.

This delay of the feedback is consistent with biologically plausible scenarios. Simultaneously, having a delay for the feedback prevents $\mathcal{N}$ from passing on the teacher value as output without first producing a prediction on its own.

Implementation: We considered a LSNN $\mathcal{N}$ consisting of 180 regularly firing neurons (population R) and 120 adapting neurons (population A) with a spread of adaptation time constants sampled uniformly between 1 and 1000 ms and with full connectivity. Sparse connectivity in conjunction with rewiring did not improve performance in this case. All neurons in the LSNN received input from a population X of 300 external input neurons. A linear readout received inputs from all neurons in R and A. The LSNN received a stream of 3 types of external inputs (see top row of Fig. 3.2D): 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), all 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. 3.2D), where the weights for this linear readout were trained, like all weights inside the LSNN, 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 target network 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). 500 of these input pairs and targets were used as training data, and presented one per step to the LSNN during the episode, where each step lasted 20 ms. LSNN parameters were updated using BPTT to minimize the mean squared error between the LSNN output and the target in the training set, using gradients computed over batches of 10 such episodes, 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 LSNN 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 LSNN 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 short-term memory and dynamics. See the suppl. for further details.

Results: Most of the functions that are computed by TNs from the class $\mathcal{F}$ are nonlinear, as illustrated in Fig. 3.2G for the case of inputs (x_1, x_2) with $x_1 = x_2$. Hence learning the input/output behaviour of any such TN with biologically realistic local plasticity mechanisms presents a daunting challenge for a SNN. Fig. 3.2C shows that after a few thousand training iterations in the outer loop, the LSNN achieves low 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 500 pairs of inputs and target outputs, see orange curve in Fig. 3.2C,E. In view of the fact that each TN is defined by 40 parameters, it comes at some surprise that the resulting network learning algorithm of the LSNN for learning the input/output behaviour of a new TN produces in general a good approximation of the TN after just 5 to 20 trials, where in each trial one randomly drawn labelled example is presented. One sample of a generic learning process is

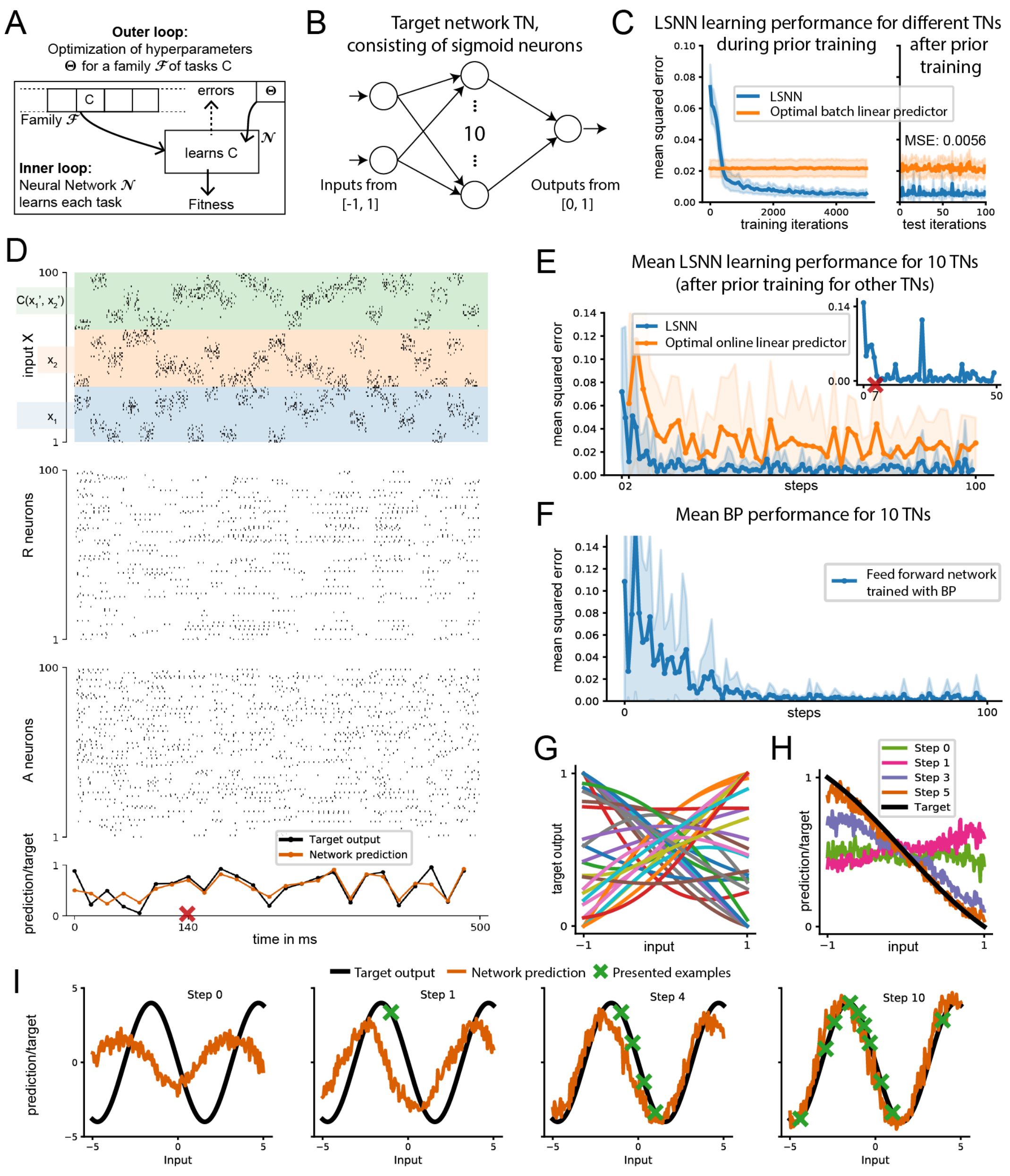


Fig. 3.2: LSNNs learn to learn from a teacher. A L2L scheme for an SNN $\mathcal{N}$. **B** Architecture of the two-layer feed-forward target networks (TNs) used to generate nonlinear functions for the LSNN to learn; weights and biases were randomly drawn from [-1,1]. **C** Performance of the LSNN in learning a new TN during (left) and after (right) training in the outer loop of L2L. Performance is compared to that of an optimal linear predictor fitted to the batch of all 500 experiments for a TN. **D** Network input (top row, only 100 of 300 neurons shown), internal spike-based processing with low firing rates in the populations R and A (middle rows), and network output (bottom row) for 25 trials of 20 ms each. **E** Learning performance of the LSNN for 10 new TNs. Performance for a single TN is shown as insert, a red cross marks step 7 after which output predictions became very good for this TN. The spike raster for this learning process is the one depicted in C. Performance is compared to that of an optimal linear predictor, which, for each example, is fitted to the batch of all preceding examples. **F** Learning performance of BP for the same 10 TNs as in D, working directly on the ANN from A, with a prior for small weights. **G** Sample input/output curves of TNs on a 1D subset of the 2D input space, for different weight and bias values. **H** These curves are all fairly smooth, like the internal models produced by the LSNN while learning a particular TN. **I** 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.

shown in Fig. 3.2D. Each sequence of examples evokes an internal model that is stored in the short-term memory of the LSNN. Fig. 3.2H shows the fast evolution of internal models of the LSNN for the TN during the first trials (visualized for a 1D subset of the 2D input space). We make the current internal model of the LSNN visible by probing its prediction $C(x_1, x_2)$ for hypothetical new inputs for evenly spaced points (x_1, x_2) 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). One sees that the internal model of the LSNN 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 LSNN 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, since Fig. 3.2E and F show that the LSNN 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 in Fig. 2A: BP with a prior that favored small weights and biases (see end of Sec. 3 in suppl.). These results suggest that L2L is able to install some form of prior knowledge about the task in the LSNN. We conjectured that the LSNN 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 sinus functions with arbitrary phase and amplitudes between 0.1 and 5. Fig. 3.2I shows that the LSNN also acquired an internal model for sinus functions (made visible analogously as in Fig. 3.2H) in this setup from training in the outer loop. 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 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, since we are not aware of previously documented methods for installing structural priors for online learning of a recurrent network of spiking neurons.

3.6 LSNNs learn-to-learn from reward

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

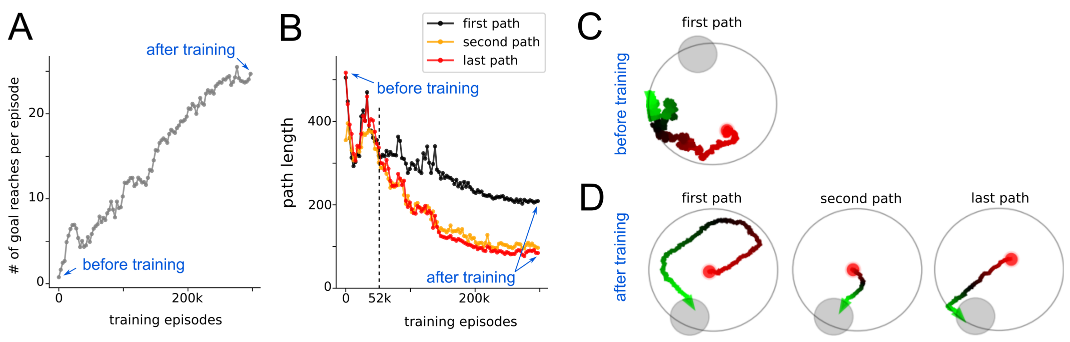


Fig. 3.3: Meta-RL results for an LSNN. A, B Performance improvement during training in the outer loop. **C, D** Samples of navigation paths produced by the LSNN before and after this training. Before training, the agent performs a random walk (**C**). In this example it does not find the goal within 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.

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.

Task: 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.

Implementation: 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 made it already challenging to move along a smooth path, or to stay within a safe distance from the wall. The agent received information 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 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 1200 ms), the network was rewired with a fixed connectivity of 20%. The resulting network diagram and spike raster is shown in Suppl. Fig. 1.

Results: The network behaviour before, during, and after L2L optimization is shown in Fig. 3.3. Fig. 3.3A shows that a large number of training episodes finally provides significant improvements. With a close look at Fig. 3.3B, 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. 3.3D that both pieces of abstract knowledge had been discovered by the agent. The first path in Fig. 3.3D shows that the agent exploits 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 not seen capabilities of sparsely firing RSNNs to extract abstract knowledge from experimentation, and to use it in clever ways for controlling behaviour.

3.7 Discussion

We have demonstrated that deep learning provides a useful new tool for the investigation of networks of spiking neurons: It allows us to create architectures and learning algorithms for RSNNs with enhanced computing and learning capabilities. In order to demonstrate this, we adapted BPTT so that it works efficiently for RSNNs, and can be combined with a biologically inspired synaptic rewiring method (DEEP R). We have shown in section 3.4 that this method allows us to create sparsely connected RSNNs that approach the performance of LSTM networks on common benchmark tasks for the classification of spatio-temporal patterns (sequential MNIST and TIMIT). This qualitative jump in the computational power of RSNNs was supported by the introduction of adapting neurons into the model. Adapting neurons introduce a spread of longer time constants into RSNNs, as they do in the neocortex according to Allen Institute: Cell Types Database, 2018. We refer to the resulting variation of the RSNN model as LSNNs, because of the resulting longer short-term memory capability. This form of short-term memory is of particular interest from the perspective of energy efficiency of SNNs, because it stores and transmits stored information through non-firing of neurons: A neuron that holds information in its increased firing threshold tends to fire less often.

We have shown in Fig. 3.2 that an application of deep learning (BPTT and DEEP R) in the outer loop of L2L provides a new paradigm for learning of nonlinear input/output mappings by a RSNN. This learning task was thought to require an implementation of BP in the RSNN. We have shown that it requires no BP, not even changes of synaptic weights. Furthermore we have shown that this new form of network learning enables RSNNs, after suitable training with similar learning tasks in the outer loop of L2L, to learn a new task from the same class substantially faster. The reason is that the prior deep learning has installed abstract knowledge (priors) about common properties of these learning tasks in the RSNN. To the best of our knowledge, transfer learning capabilities and the use of prior knowledge (see Fig. 3.2I) have previously not been demonstrated for SNNs. Fig 3.3 shows that L2L also embraces the capability of RSNNs to learn from rewards (meta-RL). For example, it enables a RSNN – 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. Here, for the sake of simplicity, we considered only the more common case when all synaptic weights are determined by the outer loop of L2L. But similar results arise when only some of the synaptic weights are learnt in the outer loop, while other synapses employ local synaptic plasticity rules to learn the current task Subramoney et al., 2018.

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 Fig. 3.1-3.3 all compute and learn with a brain-like sparse firing activity, quite different from a SNN that operates with rate-codes. In addition, these RSNNs present new functional uses of short-term memory that go far beyond remembering a preceding input as in (Mongillo et al., 2008), and suggest new forms of activity-silent memory (Stokes, 2015).

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 through non-firing.

Acknowledgments

This research/project was supported by the HBP Joint Platform, funded from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 720270 (Human Brain Project SGA1) and under the Specific Grant Agreement No. 785907 (Human Brain Project SGA2). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro P6000 GPU used for this research. Research leading to these results has in parts been carried out on the Human Brain Project PCP Pilot Systems at the Jülich Supercomputing Centre, which received co-funding from the European Union (Grant Agreement No. 604102). We gratefully acknowledge Sandra Diaz, Alexander Peyser and Wouter Klijn from the Simulation Laboratory Neuroscience of the Jülich Supercomputing Centre for their support. The computational results presented have been achieved in part using the Vienna Scientific Cluster (VSC).

Chapter 4

Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets

Contents

The way how recurrently connected networks of spiking neurons in the brain acquire powerful information processing capabilities through learning has remained a mystery. This lack of understanding is linked to a lack of learning algorithms for recurrent networks of spiking neurons (RSNNs) that are both functionally powerful and can be implemented by known biological mechanisms. Since RSNNs are simultaneously a primary target for implementations of brain-inspired circuits in neuromorphic hardware, this lack of algorithmic insight also hinders technological progress in that area. The gold standard for learning in recurrent neural networks in machine learning is back-propagation through time (BPTT), which implements stochastic gradient descent with regard to a given loss function. But BPTT is unrealistic from a biological perspective, since it requires a transmission of error signals backwards in time and in space, i.e., from post- to presynaptic neurons. We show that an online merging of locally available information during a computation with suitable top-down learning signals in real-time provides highly capable approximations to BPTT. For tasks where information on errors arises only late during a network computation, we enrich locally available information through feedforward eligibility traces of synapses that can easily be computed in an online manner. The resulting new generation of learning algorithms for recurrent neural networks provides a new understanding of network learning in the brain that can be tested experimentally. In addition, these algorithms provide efficient methods for on-chip training of RSNNs in neuromorphic hardware.

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

> Guillaume Bellec*, Franz Scherr*, Darjan Salaj, Elias Hajek, Robert Legenstein, Wolfgang Maass (2019). "Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets." *Arxiv*.

To this study, GB and FS contributed as first authors. The theory of e-prop was first discovered by GB in a particular case but most of the theoretical work was developed jointly with FS. All authors contributed to the design of the experiments. The experiments on e-prop 1 were conducted by DS, EH and GB. The experiments on e-prop 2 were conducted by FS. The experiments on e-prop 3 were conducted by GB and DS. All authors contributed to the writing of the manuscript, in particular, introduction and conclusion were written by WM and the result sections were mainly written by WM, RL and GB.

4.1 Introduction

A characteristic property of networks of neurons in the brain is that they are recurrently connected: „the brain is essentially a multitude of superimposed and ever-growing loops between the input from the environment and the brain's outputs" (Buzsaki, 2006). In fact, already (Lorente de Nó, 1938) had proposed that synaptic loops were the basic circuits of the central nervous system, and a large body of experimental work supports this view (Kandel et al., 2000). Recurrent loops of synaptic connections occur both locally within a lamina of a cortical microcircuit, between their laminae, between patches of neural tissue within the same brain area, and between different brain areas. Hence the architecture of neural networks in the brain is fundamentally different from that of feedforward deep neural network models that have gained high attention because of their astounding capability in machine learning (LeCun et al., 2015).

Recurrently connected neural networks tend to provide functionally superior neural network architectures for tasks that involve a temporal dimension, such as video prediction, gesture recognition, speech recognition, or motor control. Since the brain has to solve similar tasks, and even transforms image recognition into a temporal task via eye-movements, there is a clear functional reason why the brain employs recurrently connected neural networks. In addition, recurrent networks enable the brain to engage memory on several temporal scales, and to represent and continuously update internal states as well as goals. Furthermore the brain is a powerful prediction machine that learns through self-supervised learning to predict the consequences of its actions and of external events. In fact, predictions provide the brain with a powerful strategy for compensating the relative slowness of its sensory feedback.

The computational function of recurrently connected neural networks in the brain arises from a combination of nature and nurture that has remained opaque. In particular, it has remained a mystery how recurrent networks of spiking neurons (RSNNs) can learn. Recurrent networks of artificial neurons are commonly trained in machine learning through BPTT. BPTT can not only be used to implement supervised learning, but – with a suitably defined loss function E – self-supervised, unsupervised and reward based learning. Unfortunately BPTT requires a physically unrealistic propagation of error signals backwards in time. This feature also thwarts an efficient implementation in neuromorphic hardware. It even hinders an efficient implementation of BP or BPTT on GPUs and other standard computing hardware: „backpropagation results in locking – the weights of a network module can only be updated after a full forward propagation of data, followed by loss evaluation, and then finally after waiting for the backpropagation of error gradients" (Czarnecki et al., 2017). Locking is an issue of particular relevance for applications of BPTT to recurrent neural networks, since this amounts to applications of backpropagation to the unrolled recurrent network, which easily becomes several thousands of layers deep.

We show that BPTT can be represented by a sum of products based on a new factorization or errors gradients with regards to the synaptic weights θ_{ji}. The error gradient is represented here as a sum over t of an eligibility trace e_{ji}^t until time t - which is independent from error signals - and a learning signal L_j^t that reaches this synapse at time t, see equation (4.1). This can be interpreted as on online merging for every time step t of eligibility traces and learning signals. Because of the prominent role which forward propagation of eligibility traces play in the resulting approximations to BPTT we refer to these new algorithms as ***e-prop***.

The key problem for achieving good learning results with eligibility traces is the online production of suitable learning signals that gate the update of the synaptic weight at time t. In order to achieve the full learning power of BPTT, this learning signal would still have to be complex and questionable from a biological perspective. But several biologically plausible approximations of such online learning signals turn out to work surprisingly well, especially for tasks that recurrent networks of neurons in the brain need to solve.

There exists an abundance of experimental data on learning- or error signals in the brain. A rich literature documents the error-related negativity (ERN) that is recorded by EEG-electrodes from the human brain. The ERN has the form of a sharp negative-going deflection that accompanies behavioral errors, for example in motor control. Remarkable is that the ERN appears very fast, even before direct evidence of a behavioral error becomes accessible through sensory feedback (see e.g. Fig. 4 in (MacLean et al., 2015)), suggesting that it employs an internal error prediction network. Furthermore the amplitude of the ERN correlates with improved performance on subsequent trials ((Gehring et al., 1993), see also the review in (Buzzell et al., 2017)). These results suggest that the ERN is in fact a signal that gates learning. The data of (Buzzell et al., 2017) also shows that the ERN is generated by a distributed system of brain areas, in which posterior cingulate cortex, dorsal anterior cingulate, and parietal cortex assume dominant roles from early stages of development on. Furthermore, error-related activity from additional brain areas – insula, orbitofrontal cortex, and inferior frontal gyrus – increases with age. These experimental data suggest that error signals in the human brain are partially innate, but are complemented and refined during development.

The precise way how these error signals gate synaptic plasticity in the brain is unknown. One conjectured mechanism involves top-down disinhibition of dendrites and neurons, e.g. by activating VIP-interneurons in layer 1, which inhibit somatostatin-positive (SOM+) inhibitory neurons. Hence the activation of VIP neurons temporarily removes the inhibitory lock which SOM+ neurons hold on activity and plasticity in distal dendrites of pyramidal cells (Pi et al., 2013). Another pathway for top-down regulation of synaptic plasticity involves cholinergic activation of astrocytes (Sugihara et al., 2016). Furthermore the cerebellum is known to play a prominent role in the processing of error signals and gating plasticity (D'Angelo et al., 2016) Most importantly, the neuromodulator dopamine plays an essential role in the control of learning, in particular also for learning of motor skills (Hosp et al., 2011). Experimental data verify that neuromodulators interact with local eligibility traces in gating synaptic plasticity, see (Gerstner et al., 2018) for a review. Of interest for the context of this paper is also the recent discovery that dopaminergic neurons in the mid-brain do not emit a uniform global signal, but rather a multitude of spatially organized signals for different populations of neurons (Engelhard et al., 2019). This distributed architecture of the error-monitoring system in the brain is consistent with the assumption that local populations of neurons receive different learning signals that have been shaped during evolution and development.

Error signals in the brain are from the functional perspective reminiscent of error signals that have turned out to alleviate the need for backprogation of error signals in feedforward neural networks. A particularly interesting version of such signals is called broadcast alignment (BA) in (Samadi et al., 2017) and direct feedback alignment in (Nøkland, 2016). These error signals are sent directly from the output stage of the network to each layer of the feedforward network. If one applies this broadcast alignment idea to the unrolled feedforward version of a recurrent network, one still runs into the problem that an error broadcast to an earlier time-slice or layer would have to go backwards in time. We present a simple method where this can be avoided, which we call *e-prop 1*.

Besides BA we explore in this paper two other methods for generating learning signals that provide – in combination with eligibility traces – powerful alternatives to BPTT. In *e-prop 2* we apply the Learning-to-Learn (L2L) framework to train separate neural networks – called error modules – to produce suitable learning signals for large families of learning tasks. But in contrast to the L2L approach of (J. X. Wang et al., 2016) and (Duan et al., 2016) we allow the recurrent neural network to modify its synaptic weights for learning a particular task. Only the synaptic weights within the error module are determined on the larger time scale of the outer loop of L2L (see the scheme in Figure 4.3). We show that this approach opens new doors for learning in recurrent networks of spiking neurons, enabling for example one-shot learning of pattern generation. Our third method, *e-prop 3*, employs the synthetic gradient approach of (Jaderberg et al., 2016) and (Czarnecki et al., 2017). We show that eligibility traces substantially enhance the power of synthetic gradients, surpassing in some cases even the performance of full BPTT for artificial neural networks. Altogether the *e-prop* approach suggests that a rich reservoir of algorithmic improvements of network learning waits to be discovered, where one employs dedicated modules and processes for generating learning signals that enable learning without backpropagated error signals. In addition this research is likely to throw light on the functional role of the complex distributed architecture of brain areas that are involved in the generation of learning signals in the brain.

These *e-prop* algorithms have an attractive feature from the theoretical perspective: They can be viewed – and analyzed – as approximations to a theoretically ideal: stochastic gradient descent, or BPTT. *E-prop* algorithms are also of particular interest from the perspective of understanding learning in RSNNs of the brain. They tend to provide better learning performance for RSNNs than previously known methods. In addition, in contrast to most of the previously used methods, they do not require biologically unrealistic ingredients. In fact, it turns out that network learning with *e-prop* provides a novel understanding of refined STDP rules (Clopath et al., 2010) from a network learning perspective, that had been proposed in order to fit detailed experimental data on local synaptic plasticity mechanisms (Ngezahayo et al., 2000; Sjöström et al., 2001; Nevian and Sakmann, 2006).

In addition, *e-prop* provides a promising new approach for implementing on-chip learning in RSNNs that are implemented in neuromorphic hardware, such as Brainscales (Schemmel et al., 2010), SpiNNaker (Furber et al., 2014) and Loihi (Davies et al., 2018). Backpropagation of error signals in time as well as locking are formidable obstacles for an efficient implementation of BPTT on a neuromorphic chip. These obstacles are alleviated by the *e-prop* method.

Synaptic plasticity algorithms involving eligibility traces and gating factors have been reviewed for reinforcement learning in (Frémaux and Gerstner, 2016), see (Gerstner et al., 2018) for their relationships to data. We re-define eligibility traces for the different context of gradient descent learning. Eligibility traces are used in classical reinforcement learning theory (Sutton and Barto, 1998) to relate the (recent) history of network activity to later rewards. This reinforcement learning theory inspired our approach on a conceptual level, but the details of the mathematical analysis become quite different, since we address a different problem: how to approximate error gradients in recurrent neural networks.

We will derive in the first section of Results the basic factorization equation (4.1) that underlies our *e-prop* approach. We then discuss applications of *e-prop 1* to RSNNs with a simple BA-like learning signal. In the subsequent section we will show how an application of L2L in *e-prop 2* can improve the learning capability of RSNNs. Finally, we show in the last subsection on *e-prop 3* that adding eligibility traces to the synthetic gradient approach of (Jaderberg et al., 2016) and (Czarnecki et al., 2017) improves learning also for recurrent networks of artificial neurons.

4.2 Results

Network models The learning rules that we describe can be applied to a variety of recurrent neural network models: Standard RSNNs consisting of leaky integrate-and-fire (LIF) neurons, LSNNs (Long short term memory Spiking Neural Networks) that also contain adaptive spiking neurons (Bellec et al., 2018c), and networks of LSTM (long short-term memory) units (Hochreiter and Schmidhuber, 1997). LSNN were introduced to capture parts of the function of LSTM network in biologically motivated neural network models. In order to elucidate the link to biology, we focus in the first two variants of *e-prop* on the LIF neuron model (see Figures 4.1, 4.2 and 4.3). The LIF model is a simplified model of biological neurons: each neuron integrates incoming currents into its membrane potential, and as soon as the membrane potential crosses a threshold from below, the neuron "spikes" and a current is sent to subsequent neurons. Mathematically, the membrane potential is a leaky integrator of a weighted sum of the input currents, and the spike is a binary variable that becomes non-zero when a spike occurs (see equation (D.10) and (D.11) in Methods). To enrich the temporal processing capability of the network (see Figure 4.2), a portion of the neurons in an LSNN have adaptive firing thresholds. The dynamics of the adaptive thresholds is defined in equation (D.16) in Methods. We also applied a third variant of *e-prop 3* to LSTM networks to show that *e-prop* algorithms can be competitive on machine learning benchmarks (see Figure 4.4).

To describe the common core of these *e-prop* algorithms, all network models are subsumed under a general formalism. We assume that each neuron j is at time t in an internal state $\boldsymbol{s}_j^t \in \mathbb{R}^d$ and emits an observable state z_j^t. We also assume that $\boldsymbol{s}_j^t$ depends on the other neurons only through the vector $\boldsymbol{z}^{t-1}$ of observable states of all neurons in the network. Then, the network dynamics takes for some functions M and f the form: $\boldsymbol{s}_j^t = M(\boldsymbol{s}_j^{t-1}, \boldsymbol{z}^{t-1}, \boldsymbol{x}^t, \boldsymbol{\theta})$ and $z_j^t = f(\boldsymbol{s}_j^t)$, where $\boldsymbol{\theta}$ is the vector of model parameters (in the models considered here, synaptic weights). For instance for LIF neurons with adaptive thresholds, the internal state $\boldsymbol{s}_j^t$ of neuron j is a vector of size $d = 2$ formed by the membrane voltage and the adaptive firing threshold, and the observable state $z_j^t \in \{0, 1\}$ indicates whether the neuron spikes at time t. The definition of the functions M and f defining the neuron dynamics for this model are given by equations (D.10),(D.11) and (D.16) in Methods and illustrated in Figure D.1.

Mathematical framework for *e-prop* algorithms The fundamental mathematical law that enables the *e-prop* approach is that the gradients of BPTT can be factorized as a sum of products between learning signals L_j^t and eligibility traces e_{ji}^t. We subsume here under the term eligibility trace that information which is locally

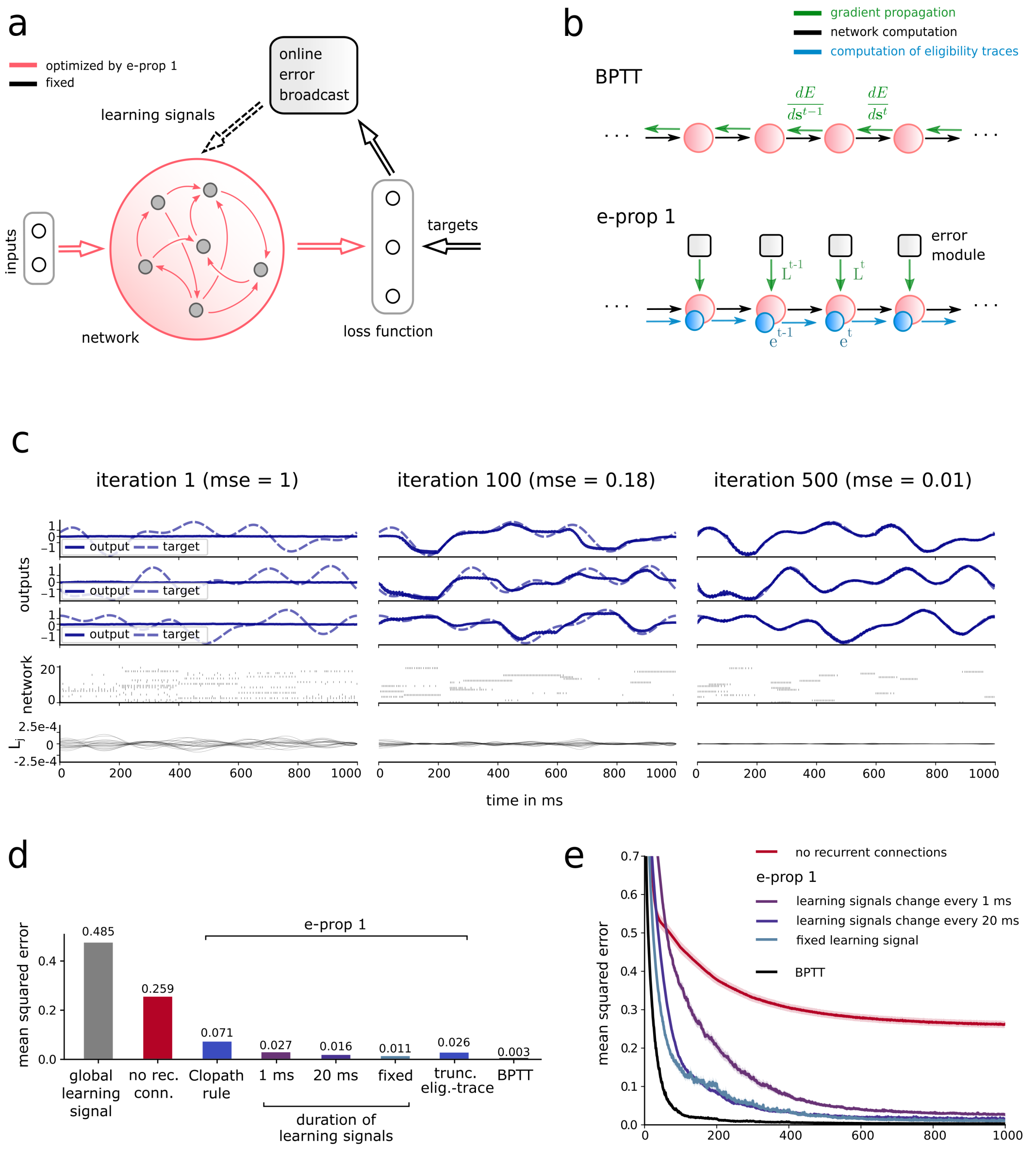


Fig. 4.1: Scheme and performance of *e-prop 1* **a**) Learning architecture for *e-prop 1*. The error module at the top sends online error signals with random weights to the network that learns. **b**) Temporal dynamics of information flows in BPTT and *e-prop* algorithms. The propagation of error signals backwards in time of BPTT is replaced in *e-prop* algorithms by an additional computation that runs forward in time: the computation of eligibility traces. **c**) Evaluation of *e-prop 1* for a classical benchmark task for learning in recurrent SNNs: Learning to generate a target pattern, extended here to the challenge to simultaneously learn to generate 3 different patterns, which makes credit assignment for errors more difficult. **d**) Mean squared error of several learning algorithms for this task. “Clopath rule” denotes a replacement of the resulting synaptic plasticity rule of *e-prop 1* by the rule proposed in (Clopath et al., 2010) based on experimental data. **e**) Evolution of the mean squared error during learning.

available at a synapse and does not depend on network performance. The online learning signals L_j^t are provided externally and could for example quantify how spiking at the current time influences current and future errors. The general goal is to approximate the gradients of the network error function E with respect to the model parameters $\boldsymbol{\theta}_{ji}$. If the error function E depends exclusively on the network spikes $E(\mathbf{z}^1, \ldots, \mathbf{z}^T)$, the fundamental observation for *e-prop* is that the gradient with respect to the weights can be factorized as follows (see Methods for a proof):

$$\frac{dE}{d\theta_{ji}} = \sum_t L_j^t \, e_{ji}^t \, . \tag{4.1}$$

We refer to L_j^t and e_{ji}^t as the learning signals and eligibility traces respectively, see below for a definition. Note that we use distinct notation for the partial derivative $\frac{\partial E(\mathbf{z}_1, \ldots, \mathbf{z}_T)}{\partial \mathbf{z}_1}$, which is the derivative of the mathematical function E with respect to its first variable $\mathbf{z}_1$, and the total derivative $\frac{dE(\mathbf{z}_1, \ldots, \mathbf{z}_T)}{d\mathbf{z}_1}$ which also takes into account how E depends on z_1 indirectly through the other variables $\mathbf{z}_1, \ldots, \mathbf{z}_T$. The essential term for gradient descent learning is the total derivative $\frac{dE}{d\theta_{ji}}$. It has usually been computed with BPTT (Werbos, 1990) or RTRL (Williams and Zipser, 1989).

Eligibility traces The intuition for eligibility traces is that a synapse remembers some of its activation history while ignoring inter neuron dependencies. Since the network dynamics is formalized through the equation $\boldsymbol{s}_j^t = M(\boldsymbol{s}_j^{t-1}, \boldsymbol{z}^{t-1}, \boldsymbol{x}^t, \boldsymbol{\theta})$, the internal neuron dynamics isolated from the rest of the network is described by $D_j^{t-1} = \frac{\partial M}{\partial \boldsymbol{s}_j^{t-1}}(\boldsymbol{s}_j^{t-1}, \boldsymbol{z}^{t-1}, \boldsymbol{x}^t, \boldsymbol{\theta}) \in \mathbb{R}^{d \times d}$ (recall that d, is the dimension of internal state of a single neuron; $d = 1$ or 2 in this paper). Considering the partial derivative of the state with respect to the synaptic weight $\frac{\partial M}{\partial \boldsymbol{\theta}_{ji}}(\boldsymbol{s}_j^{t-1}, \boldsymbol{z}^{t-1}, \boldsymbol{x}^t, \boldsymbol{\theta}) \in \mathbb{R}^d$ (written $\frac{\partial \boldsymbol{s}_j^t}{\partial \theta_{ji}}$ for simplicity), we formalize the mechanism that retains information about the previous activity at the synapse $i \to j$ by the eligibility vector $\boldsymbol{\epsilon}_{ji}^t \in \mathbb{R}^d$ defined with the following iterative formula:

$$\boldsymbol{\epsilon}_{ji}^t \;=\; D_j^{t-1} \cdot \boldsymbol{\epsilon}_{ji}^{t-1} + \frac{\partial \boldsymbol{s}_j^t}{\partial \theta_{ji}} \, , \tag{4.2}$$

where $\cdot$ is the dot product. Finally, this lead to the eligibility trace which is the scalar product between this vector and the derivative $\frac{\partial f}{\partial \boldsymbol{s}_j^t}(\boldsymbol{s}_j^t)$ (denoted $\frac{\partial z_j^t}{\partial \mathbf{s}_j^t}$ for simplicity) which captures how the existence of a spike z_j^t depends on the neuron state $\boldsymbol{s}_j^t$:

$$e_{ji}^t \;=\; \frac{\partial z_j^t}{\partial \mathbf{s}_j^t} \cdot \boldsymbol{\epsilon}_{ji}^t \, . \tag{4.3}$$

In practice for LIF neurons the derivative $\frac{\partial z_j^t}{\partial \boldsymbol{s}_j^t}$ is ill-defined due to the discontinuous nature of spiking neurons. As done in (Bellec et al., 2018c) for BPTT, this derivative is replaced in simulations by a simple nonlinear function of the membrane voltage h_j^t that we call the pseudo-derivative (see Methods for details). The resulting eligibility traces e_{ji}^t for LIF neurons are the product of a the post synaptic pseudo derivative h_j^t with the trace $\hat{z}_i^t$ of the presynpatic spikes (see equation (D.12) in Methods). For adaptive neurons in LSNNs and for LSTM units the computation of eligibility traces becomes less trivial, see equations (D.17) and (D.18). But they can still be computed in an online manner forward in time, along with the network computation. We show later that the additional term arising for LSNNs in the presence of threshold adaptation holds a crucial role when working memory has to be engaged in the tasks to be learnt.

Note that in RTRL for networks of rate-based (sigmoidal) neurons (Williams and Zipser, 1989), the error gradients are computed forward in time by multiplying the full Jacobian $\boldsymbol{J}$ of the network dynamics with the tensor $\frac{d\boldsymbol{s}_k^t}{d\theta_{ji}}$ that computes the dependency of the state variables with respect to the parameters: $\frac{d\boldsymbol{s}_k^t}{d\theta_{ji}} = \sum_{k'} \boldsymbol{J}_{kk'}^t \cdot \frac{d\boldsymbol{s}_{k'}^{t-1}}{d\theta_{ji}} + \frac{\partial \boldsymbol{s}_k^t}{\partial \theta_{ji}}$ (see equation (12) in (Williams and Zipser, 1989)). Denoting with n the number of neurons, this requires $O(n^4)$ multiplications, which is computationally prohibitive in simulations, whereas BPTT or network simulation requires only $O(n^2)$ multiplications. In *e-prop*, the eligibility traces are $n \times n$ matrices which is one order smaller than the tensor $\frac{d\boldsymbol{s}_k^t}{d\theta_{ji}}$, also D_j^t are $d \times d$ matrices which are restrictions of the full Jacobian $\boldsymbol{J}$ to the neuron specific dynamics. As a consequence, only $O(n^2)$ multiplications are required in the forward propagation of eligibility traces, their computation is not more costly than BPTT or simply simulating the network.

Theoretically ideal learning signals To satisfy equation (4.1), the learning signals $L_j^t \in \mathbb{R}^d$ can be defined by the following formula:

$$L_j^t \overset{\text{def}}{=} \frac{dE}{dz_j^t} \,. \tag{4.4}$$

Recall that $\frac{dE}{dz_j^t}$ is a total derivative and quantifies how much a current change of spiking activity might influence future errors. As a consequence, a direct computation of the term L_j^t needs to back-propagate gradients from the future as in BPTT. However we show that *e-prop* tends to work well if the ideal term L_j^t is replaced by an online approximation $\widehat{L}_j^t$. In the following three sections we consider three concrete approximation methods, that define three variants of *e-prop*. If not clearly stated otherwise, all the resulting gradient estimations described below can be computed online and depend only on quantities accessible within the neuron j or the synapse $i \to j$ at the current time t.

Synaptic plasticity rules that emerge from this approach The learning algorithms *e-prop 1* and *e-prop 2* that will be discussed in the following are for networks of spiking neurons. Resulting local learning rules are very similar to previously proposed and experimentally supported synaptic plasticity rules. They have the general form (learning signal) $\times$ (postsynaptic term) $\times$ (presynaptic term) as previously proposed 3-factor learning rules (Frémaux and Gerstner, 2016; Gerstner et al., 2018). The general form is given in equation (D.13), where h_j^t denotes a postsynaptic term and the last factor $\hat{z}_i^{t-1}$ denotes the presynaptic term (D.12). These last two terms are similar to the corresponding terms in the plasticity rule of (Clopath et al., 2010). It is shown in Figure 4.1d that one gets very similar results if one replaces the rule (D.14) that emerges from our approach by the Clopath rule.

The version (4.5) of this plasticity rule for *e-prop 1* contains the specific learning signal that arises in broadcast alignment as first factor. For synaptic plasticity of adapting neurons in LSNNs the last term, the eligibility trace, becomes a bit more complex because it accumulates information over a longer time span, see equation (D.18). The resulting synaptic plasticity rules for LSTM networks are given by equation (D.28).

4.2.1 *E-prop 1*: Learning signals that arise from broadcast alignment

A breakthrough result for learning in feedforward deep neural networks was the discovery that a substantial portion of the learning power of backprop can be captured if the backpropagation of error signals through chains of layers is replaced by layer specific direct error broadcasts, that consists of a random weighted sum of the errors that are caused by the network outputs; typically in the last layer of the network (Samadi et al., 2017; Nøkland, 2016). This heuristic can in principle also be applied to the unrolled version of a recurrent neural network, yielding a different error broadcast for each layer of the unrolled network, or equivalently, for each time-slice of the computation in the recurrent network. This heuristic would suggest to send to each time slice error broadcasts that employ different random weight matrices. We found that the best results can be achieved if one chooses the same random projection of output errors for each time slice (see Figure 4.1d and e).

Definition of *e-prop 1*: *E-prop 1* defines a learning signal that only considers the instantaneous error of network outputs and ignores the influence of the current activity on future errors. As justified in Methods, this means that the approximation of the learning signal $\widehat{L}_j^t$ is defined by replacing the total error derivative $\frac{dE}{dz_j^t}$ with the partial derivative $\frac{\partial E}{\partial z_j^t}$. Crucially, this replacement makes it possible to compute the learning signal in real-time, whereas the total derivative needs information about future errors which should be back-propagated through time for an exact computation.

To exhibit an equation that summarizes the resulting weight update, we consider a network of LIF neurons and output neurons formalized by k leaky readout neurons y_k^t with decay constant κ. If E is defined as the squared error between the readouts y_k^t and their targets $y_k^{*,t}$, and the weight updates are implemented with gradient descent and learning rate η, this yields (the proof and more general formula (D.35) are given in Methods):

$$\Delta\theta_{ji}^{\mathrm{rec}} = \eta \sum_t \Big(\sum_k \theta_{kj}^{\mathrm{out}} (y_k^{*,t} - y_k^t) \Big) \sum_{t' \leq t} \kappa^{t-t'} h_j^{t'} \hat{z}_i^{t'-1} , \tag{4.5}$$

where $h_j^{t'}$ is a function of the post-synaptic membrane voltage (the pseudo-derivative, see Methods) and $\hat{z}_i^{t'}$ is a trace of preceding pre-synaptic spikes with a decay constant α. This is a three-factor learning rule of a type that is commonly used to model experimental data (Gerstner et al., 2018). However a less common feature is that, instead of a single global error signal, learning signals are neuron-specific weighted sums of different signed error signals arising from different output neurons k. For a more complex neuron model such as LIF with adaptive thresholds, which are decisive for tasks involving working memory (Bellec et al., 2018c), the eligibility traces are given in equation (D.18) and the learning rule in equation (D.35).

To model biology, a natural assumption is that synaptic connections to and from readout neurons are realized through different neurons. Therefore it is hard to conceive that the feedback weights are exactly symmetric to the readout weights, as required for gradient descent according to equation (4.5) that follows from the theory. Broadcast alignment (Lillicrap et al., 2016) suggests the replacement of $\theta_{kj}^{\mathrm{out}}$ by a random feedback matrix. We make the same choice to define a learning rule (D.39) with mild assumptions: the learning signal is a neuron-specific random projection of signed error signals $y_k^{*,t} - y_k^t$. Hence we replaced the weights $\theta_{kj}^{\mathrm{out}}$ with random feedback weights denoted B_{jk}^{random}. Other authors (Zenke and Ganguli, 2018; Kaiser et al., 2018) have derived learning rules similar to (D.39) for feedforward networks of spiking neurons but not for recurrent ones. We test the learning performance of *e-prop 1* for two generic computational tasks for recurrent neural networks: generation of a temporal pattern, and storing selected information in working memory.

Pattern generation task 1.1 Pattern generation is an important component of motor systems. We asked whether the simple learning setup of *e-prop 1* endows RSNNs with the capability to learn to generate patterns in a supervised manner.

Task: To this end, we considered a pattern generation task which is an extension of the task used in (Nicola and Clopath, 2017). In this task, the network should autonomously generate a three-dimensional target signal for 1 s. Each dimension of the target signal is given by the sum of four sinusoids with random phases and amplitudes. Similar to (Nicola and Clopath, 2017), the network received a clock input that indicates the current phase of the pattern (see Methods).

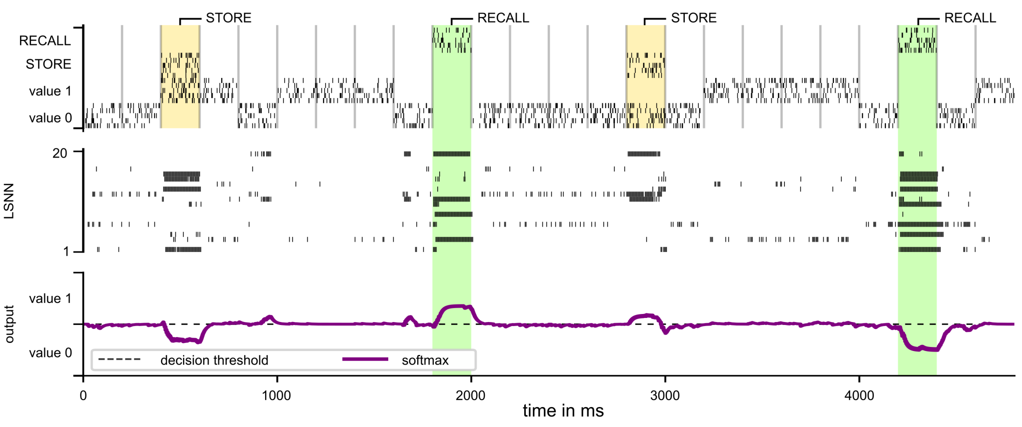
Implementation: The network consisted of 600 recurrently connected LIF neurons. All neurons in this RSNN projected to three linear readout neurons. All input, recurrent and output weights were plastic, see Figure 4.1a. A single learning trial, consisted of a 1 s simulation where the network produced a three-dimensional output pattern and gradients were computed using *e-prop 1*. Network weights were updated after each learning trial (see Methods for details).

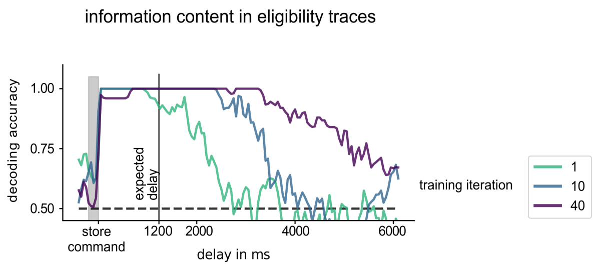
Performance: Figure 4.1c shows the spiking activity of a randomly chosen subset of 20 of 600 neurons in the RSNN along with the output of the three readout neurons after application of *e-prop 1* for 1, 100 and 500 seconds, respectively. In this representative example, the network achieved a very good fit to the target signal (normalized mean squared error 0.01). Panel d shows the averaged mean squared errors (mse) for several variants of *e-prop 1* and a few other learning methods.

As an attempt to bridge a gap between phenomenological measurements of synaptic plasticity and functional learning models, we addressed the question whether a synaptic plasticity rule that was fitted to data in (Clopath et al., 2010) could reproduce the function of the second and third factors in equation (4.5). These two terms ($\hat{z}_i^{t-1}$ and h_j^t) couple the presynaptic and postsynaptic activity in a multiplicative manner. In the model of long term potentiation fitted to data by (Clopath et al., 2010), the presynaptic term is identical but the postsynaptic term includes an additional non-linear factor depending on a filtered version of the membrane voltage. We found that a replacement of the plasticity rule of *e-prop 1* by the Clopath rule had little impact on the result, as shown in Figure 4.1d under the name "Clopath rule" (see equation (D.47) in Methods for a precise definition of this learning rule). We also asked whether these pre- and postsynaptic factors could be simplified further. When replacing the trace $\hat{z}_i^{t-1}$ and h_j^t by the binary variable z_i^{t-1}, this just caused an increase of the mse from 0.011 to 0.026. In comparison, when the network has no recurrent connections or when the learning signal is replaced by a uniform global learning signal ($B_{jk} = \frac{1}{\sqrt{n}}$ with n the number of neurons) the mse increased by one order of magnitude to 0.259 and 0.485, respectively. This indicated the importance of diverse learning signals and recurrent connections in this task.

Panel e shows that a straightforward application of BA to the unrolled RSNNs, with new random matrices for error broadcast at every ms, or every 20 ms, worked less well or converged slower. Finally, while *e-prop 1* managed to solve the task very well (Figure 4.1d), BPTT achieved an even lower mean squared error (black line in Figure 4.1e).

Store-recall task 1.2 Many learning tasks for brains involve some form of working memory. We therefore took a simple working memory task and asked whether *e-prop 1* enables an LSNN to learn this task, in spite of a substantial delay between the network decision to store information and the time when the network output makes an error.

a


b


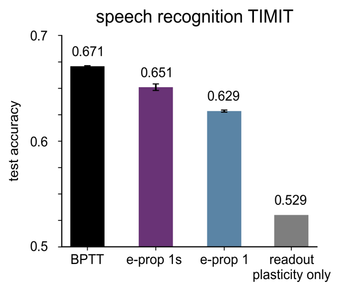
c


Fig. 4.2: Testing *e-prop 1* on the store-recall and speech recognition tasks. a) The store-recall task requires to store the value 0 or 1 that is currently provided by other input neurons when a STORE command is given, and to recall it when a RECALL command is given. It is solved with *e-prop 1* for an LSNN. **b**) Information content of eligibility traces (estimated via a linear classifier) about the bit that was to be stored during store-recall task. **c**) Training LSNNs with *e-prop 1* to solve the speech reccognition task on TIMIT dataset. *e-prop 1s* indicates the case when the feedback weights are exactly symmetric to the readout weights.

Task: The network received a sequence of binary values encoded by alternating activity of two groups of input neurons ("Value 0" and "Value 1" in Figure 4.2a, top). In addition, it received command inputs, STORE and RECALL, encoded likewise by dedicated groups of input neurons. The task for the network was to output upon a RECALL command the value that was present in the input at the time of the most recent STORE command. In other words, the network had to store a bit (at a STORE command) and recall it at a RECALL command. After a STORE, a RECALL instructions was given during each subsequent time period of length $D = 200$ ms with probability $p_{\text{command}} = \frac{1}{6}$. This resulted in an expected delay of $\frac{D}{p_{\text{command}}} = 1.2$ s between STORE and RECALL instruction. The next STORE appeared in each subsequent period of length D with probability p_{command}. We considered the task as solved when the misclassification rate on the validation set reached a value below 5%.

Implementation: We used a recurrent LSNN network consisting of 10 standard LIF neurons and 10 LIF neurons with adaptive thresholds. The input neurons marked in blue and red at the top of Figure 4.2a produced Poisson spike trains with time varying rates. An input bit to the network was encoded by spiking activity at 50 Hz in the corresponding input channels for a time period of $D = 200$ ms. STORE and RECALL instructions were analogously encoded through firing of populations of other Poisson input neurons at 50 Hz. Otherwise input neurons were silent. The four groups of input channels consisted of 25 neurons each. During a recall cue, two readouts are competing to report the network output. The readout with highest mean activation y_k^t during the recall period determines which bit value is reported. To train the network, the error function E is defined as the cross entropy between the target bit values and the softmax of the readout activations (see Methods for details). Importantly, an error signal is provided only during the duration of a RECALL command and informs about the desired change of output in that delayed time period. The network was trained with *e-prop 1*. Parameters were updated every 2.4 s during training.

Performance: Figure 4.2a shows a network trained with *e-prop 1* that stores and recalls a bit accurately. This network reached a misclassification rate on separate validation runs below 5% in 50 iterations on average with *e-prop 1*. For comparison, the same accuracy was reached within 28 iterations on average with full BPTT. We found that adaptive neurons are essential for these learning procedures to succeed: A network of 20 non-adapting LIF neurons could not solve this task, even if it was trained with BPTT.

It might appear surprising that *e-prop 1* is able to train an LSNN for this task, since the learning signal is only non-zero during a RECALL command. This appears to be problematic, because in order to reduce errors the network has to learn to handle information from the input stream in a suitable manner during a much earlier time window: during a STORE command, that appeared on average 1200 ms earlier. We hypothesized that this was made possible because eligibility traces can hold information during this delay. In this way a learning signal could contribute to the modification of the weight of a synapse that had been activated much earlier, for example during a STORE command. According to the theory (equation (D.18) in the methods), eligibility traces of adapting neurons decay with a time constant comparable to that of the threshold adaptation. To verify experimentally that this mechanism makes it possible to hold the relevant information, we first verified that the same LSNN network failed at learning the task when eligibility traces are truncated by setting $\epsilon_{ji}^t = \frac{\partial \boldsymbol{s}^t}{\partial \theta_{ji}}$. Second, we quantified the amount of information about the bit to be stored that is contained in the true eligibility traces. This information was estimated via the decoding accuracy of linear classifiers, and the results are reported in Figure 4.2b. While the neuron adaptation time constants were set to 1.2 s, we found that the decoding accuracy quickly rises above chance even for much longer delays. After 200 training iterations, the relevant bit can be decoded up to 4 s after the store signal arrived with high accuracy ($>$ 90% of the trials).

Altogether the results in Figure 4.1 and 4.2 suggest that *e-prop 1* enables RSNNs to learn the most fundamental tasks which RSNNs have to carry out in the brain: to generate desired temporal patterns and to carry out computations that involve a working memory. Pattern generation tasks were also used for testing the performance of FORCE training for RSNNs in (Nicola and Clopath, 2017). While FORCE training has not been argued to be biologically plausible because of the use of a non-local plasticity rule and the restriction of plasticity to readout neurons, *e-prop 1* only engages mechanisms that are viewed to be biologically plausible. Hence it provides a concrete hypothesis how recurrent networks of neurons in the brain can learn to solve the most fundamental tasks which such networks are conjectured to carry out. More generally, we conjecture that *e-prop 1* can solve all learning tasks that have been demonstrated to be solvable by FORCE training. However we do not want to claim that *e-prop 1* can solve all learning tasks for RSNNs that can be solved by BPTT according to (Bellec et al., 2018c). But the power of *e-prop* algorithms can be substantially enhanced by using more sophisticated learning signals than just random linear combinations of signed errors as in broadcast alignment. Several neural systems in the brain receive raw error signals from the periphery and output highly processed learning cues for individual brain areas and populations of neurons. We propose that such neural systems have been refined by evolution and development to produce learning signals that enable more powerful versions of *e-prop* algorithms, such as the ones that we will discuss in the following.

For implementations of *e-prop* algorithms in neuromorphic hardware – in order to enable efficient on-chip learning of practically relevant tasks – another reservoir of mechanisms becomes of interest that also can make use of concrete aspects of specific neuromorphic hardware. For example, in order to approximate the performance of BPTT for training LSNNs for speech recognition, a test on the popular benchmark dataset TIMIT suggests that the accuracy of *e-prop 1* (0.629) can be brought closer to that achieved by BPTT (0.671; see (Bellec et al., 2018c)) by simply using the current values of readout weights, rather than random weights, for broadcasting current error signals (we label this *e-prop 1* version *e-prop 1s*). This yields an accuracy of 0.651. As a baseline for recurrent spiking neural networks we include a result for randomly initialized LSNN where only the readout weights are optimized (see Figure 4.2c readout plasticity only, accuracy 0.529). For comparison, the best result achieved by recurrent artificial neural networks (consisting of the most complex form of LSTM units) after extensive hyperparameter search was 0.704 (Greff et al., 2017).

4.2.2 *E-prop 2*: Refined learning signals that emerge from L2L

The construction and proper distribution of learning signals appears to be highly sophisticated in the brain. Numerous areas in the human brain appear to be involved in the production of the error-related negativity and the emission of neuromodulatory signals (see the references given in the Introduction). A closer experimental analysis suggests diversity and target-specificity even for a single neuromodulator (Engelhard et al., 2019). Hence it is fair to assume that the construction and distribution of error signals in the brain has been optimized through evolutionary processes, through development, and prior learning. A simple approach for capturing possible consequences of such an optimization in a model is to apply L2L to a suitable family of learning tasks. The outer loop of L2L models opaque optimization processes that shape the distribution of error signals in the brain on the functional level. We implement this optimization by an application of BPTT to a separate error module in the outer loop of L2L. Since this outer loop is not meant to model an online learning process, we are not concerned here by the backpropagation through time that is required in the outer loop. In fact, one can expect that similar results can be achieved through an application of gradient-free optimization methods in the outer loop, but at a higher computational cost for the implementation.

It is argued in (Brea and Gerstner, 2016) that one-shot learning is one of two really important learning capabilities of the brain that are not yet satisfactorily explained by current models in computational neuroscience. We show here that *e-prop 2* explains how RSNNs can learn a new movement trajectory in a single trial. Simultaneously we show that given movement trajectories of an end-effector of an arm model can be learnt without requiring an explicitly learnt or constructed inverse model. Instead, a suitably trained error module can acquire the capability to produce learning signals for the RSNN so that the RSNN learns via *e-prop* to minimize deviations of the end-effector from the target trajectory in Euclidean space, rather than errors in "muscle space", i.e., in terms of the joint angles that are controlled by the RSNN.

Definition of *e-prop 2*: The main characteristic of *e-prop 2* is that the learning signals $\widehat{L}_j^t$ are produced by a trained error module, which is modeled as a recurrent network of spiking neurons with synaptic weights $\mathbf{\Psi}$. It receives the input $\boldsymbol{x}^t$, the spiking activity in the network $\boldsymbol{z}^t$ and target signals $\boldsymbol{y}^{*,t}$. Note that the target signal

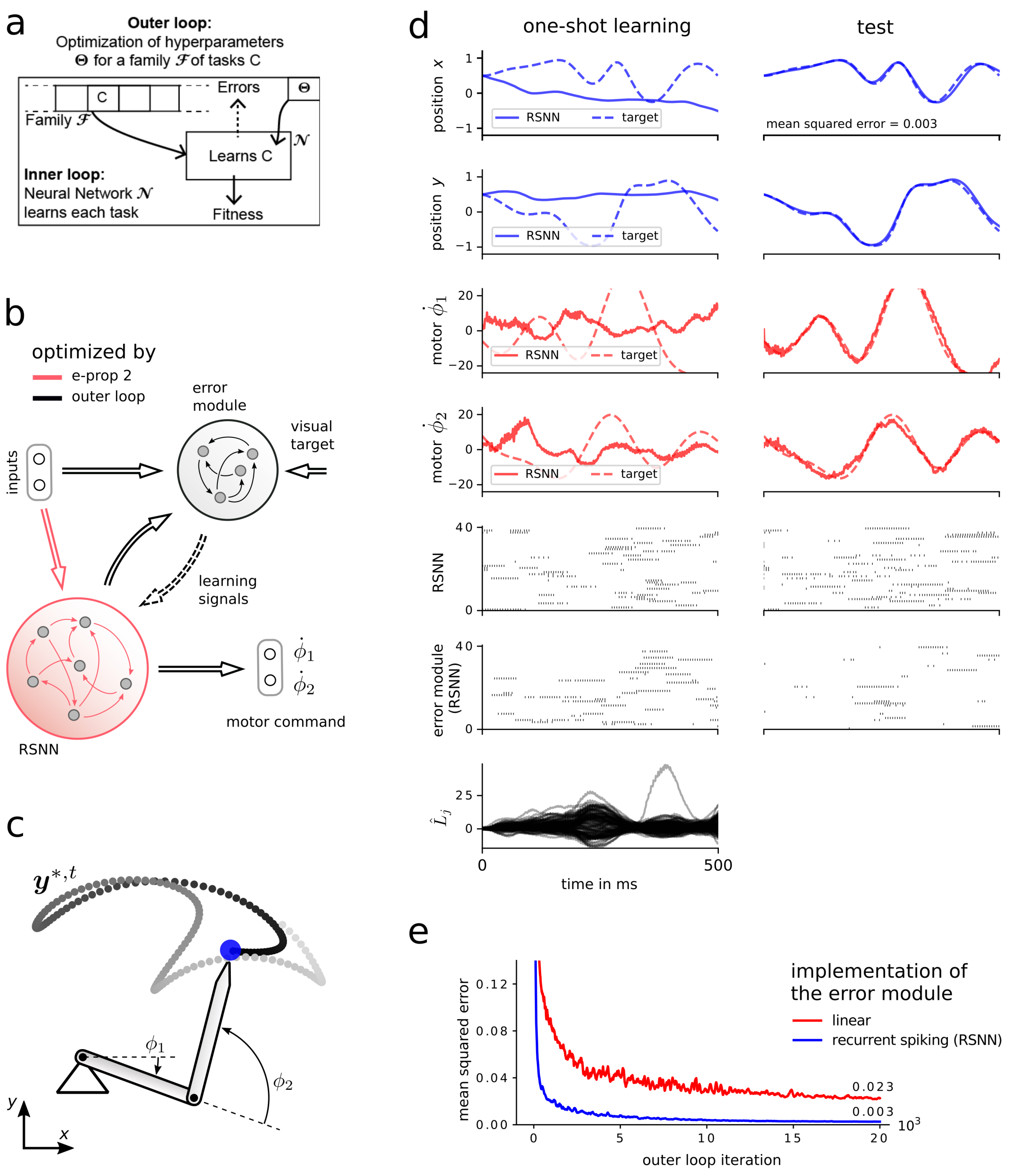


Fig. 4.3: Scheme and performance of *e-prop 2* **a**) Learning-to-Learn (LTL) scheme. **b**) Learning architecture for *e-prop 2*. In this demo the angular velocities of the joints were controlled by a recurrent network of spiking neurons (RSNN). A separate error module was optimized in the outer loop of L2L to produce suitable learning signals. **c**) Randomly generated target movements $\boldsymbol{y}^{*,t}$ (example shown) had to be reproduced by the tip of an arm with two joints. **d**) Demonstration of one-shot learning for a randomly sampled target movement. During the training trial the error module sends learning signals (bottom row) to the network. After a single weight update the target movement can be reproduced in a test trial with high precision. **e**) One-shot learning performance improved during the course of outer loop optimization. Two error module implementations were compared.

is not necessarily the target output of the network, but can be more generally a target state vector of some controlled system.

We employ the concept of Learning-to-Learn (L2L) to enable the network with its adjacent error module to solve a family $\mathcal{F}$ of one-shot learning tasks, i.e. each task C of the family $\mathcal{F}$ requires the network to learn a movement from a single demonstration. The L2L setup introduces a nested optimization procedure that consists of two loops: An inner loop and an outer loop as illustrated in Figure 4.3a. In the inner loop we consider a particular task C, entailing a training trial and a testing trial. During the training trial, the network has synaptic weights $\boldsymbol{\theta}_{\text{init}}$, it receives an input it has never seen and generates a tentative output. After a single weight update using *e-prop 2*, the network starts the testing trial with new weights $\boldsymbol{\theta}_{\text{test},C}$. It receives the same input for a second time and its performance is evaluated using the cost function $\mathcal{L}_C(\boldsymbol{\theta}_{\text{test},C})$. In the outer loop we optimize $\boldsymbol{\theta}_{\text{init}}$ and the error module parameters $\boldsymbol{\Psi}$ in order to minimize the cost $\mathcal{L}_C$ over many task instances C from the family $\mathcal{F}$. Formally, the optimization problem solved by the outer loop is written as:

$$\min_{\boldsymbol{\Psi},\boldsymbol{\theta}_{\text{init}}} \qquad \mathbb{E}_{C\sim\mathcal{F}}\left[\mathcal{L}_C(\boldsymbol{\theta}_{\text{test},C})\right] \tag{4.6}$$

$$\text{s.t.:} \qquad (\boldsymbol{\theta}_{\text{test},C})_{ji} = (\boldsymbol{\theta}_{\text{init}})_{ji} - \eta \textstyle\sum_t \hat{L}_j^t \, e_{ji}^t \tag{4.7}$$

$$(\hat{L}_j^t \text{ and } e_{ji}^t \text{ are obtained during the training trial for task } C \text{ using } \boldsymbol{\Psi} \text{ and } \theta_{\text{init}}),$$

where η represents a fixed learning rate.

One-shot learning task 2.1 It is likely that prior optimization processes on much slower time scales, such as evolution and development, have prepared many species of animals to learn new motor skills much faster than shown in Figure 4.1d. Humans and other species can learn a new movement by observing just one or a few examples. Therefore we restricted the learning process here to a single trial (one-shot learning, or imitation learning).

Another challenge for motor learning arises from the fact that motor commands $\dot{\Phi}^t$ have to be given in "muscle space" (joint angle movements), whereas the observed resulting movement $\boldsymbol{y}^t$ is given in Euclidean space. Hence an inverse model is usually assumed to be needed to infer joint angle movements that can reduce the observed error. We show here that an explicit inverse model is not needed, since its function can be integrated into the learning signals from the error module of *e-prop 2*.

Task: Each task C in the family $\mathcal{F}$ consisted of learning a randomly generated target movement $\boldsymbol{y}^{*,t}$ of the tip of a two joint arm as shown in Figure 4.3c. The task was divided into a training and a testing trial, with a single weight update in between according to equation (4.7).

Implementation: An RSNN, consisting of 400 recurrently connected LIF neurons, learnt to generate the required motor commands, represented as the angular velocities of the joints $\dot{\Phi}^t = (\dot{\phi}_1^t, \dot{\phi}_2^t)$, in order to produce the target movement. The full architecture of the learning system is displayed in Figure 4.3b. The error module consisted of 300 LIF neurons, which were also recurrently connected. The input $\boldsymbol{x}^t$ to the network was the same across all trials and was given by a clock-like signal. The input to the error module contained a copy of $\boldsymbol{x}^t$, the spiking activity $\boldsymbol{z}^t$ of the main network, as well as the target movement $\boldsymbol{y}^{*,t}$ in Euclidean space. Importantly, the error module had no access to actual errors of the produced motor commands. For outer loop optimization we viewed the learning process as a dynamical system for which we applied BPTT. Gradients were computed using batches of different tasks to approximate the expectation in the outer loop objective.

Performance: After sufficiently long training in the outer loop of L2L, we tested the learning capabilities of *e-prop 2* on a random target movement, and show in Figure 4.3d training and testing trial in the left and right column respectively. In fact, after the error module had sent learning signals to the network during the training trial, it was usually more silent during testing, since the reproduced movement was already accurate. Therefore, the network was endowed with one-shot learning capabilities by *e-prop 2*, after initial weights $\boldsymbol{\theta}_{\text{init}}$ and the error module parameters $\boldsymbol{\Psi}$ had been optimized in the outer loop.

Figure 4.3e summarizes the mean squared error between the target $\boldsymbol{y}^{*,t}$ and actual movement $\boldsymbol{y}^t$ in the testing trial (blue curve). The red curve reports the same for a linear error module. The error is reported for different stages of the outer loop optimization.

We considered also the case when one uses instead of the eligibility trace as defined in equation (D.13) just a truncated one, given by $e_{ji}^t = h_j^t z_i^{t-1}$, and found this variation to exhibit similar performance on this task (not shown). The learning performance converged to a mean squared error of 0.005 on testing trials averaged over different tasks.

Altogether we have shown that *e-prop 2* enables one-shot learning of pattern generation by an RSNN. This is apparently the first time that one-shot learning of pattern generation has been demonstrated for an RSNN. In addition, we have shown that the learning architecture for *e-prop 2* supports a novel solution to motor learning, where no separate construction or learning of an inverse model is required. More precisely, the error module can be trained to produce learning signals that enable motor learning without the presence of an inverse model. It will be interesting to see whether there are biological data that support this simplified architecture. Another interesting feature of the resulting paradigm for motor learning is that the considered tasks can be accomplished without sensory feedback about the actual trajectory of the arm movement. Instead the error module just receives efferent copies of the spiking activity of the main network, and hence implicitly also of its motor commands.

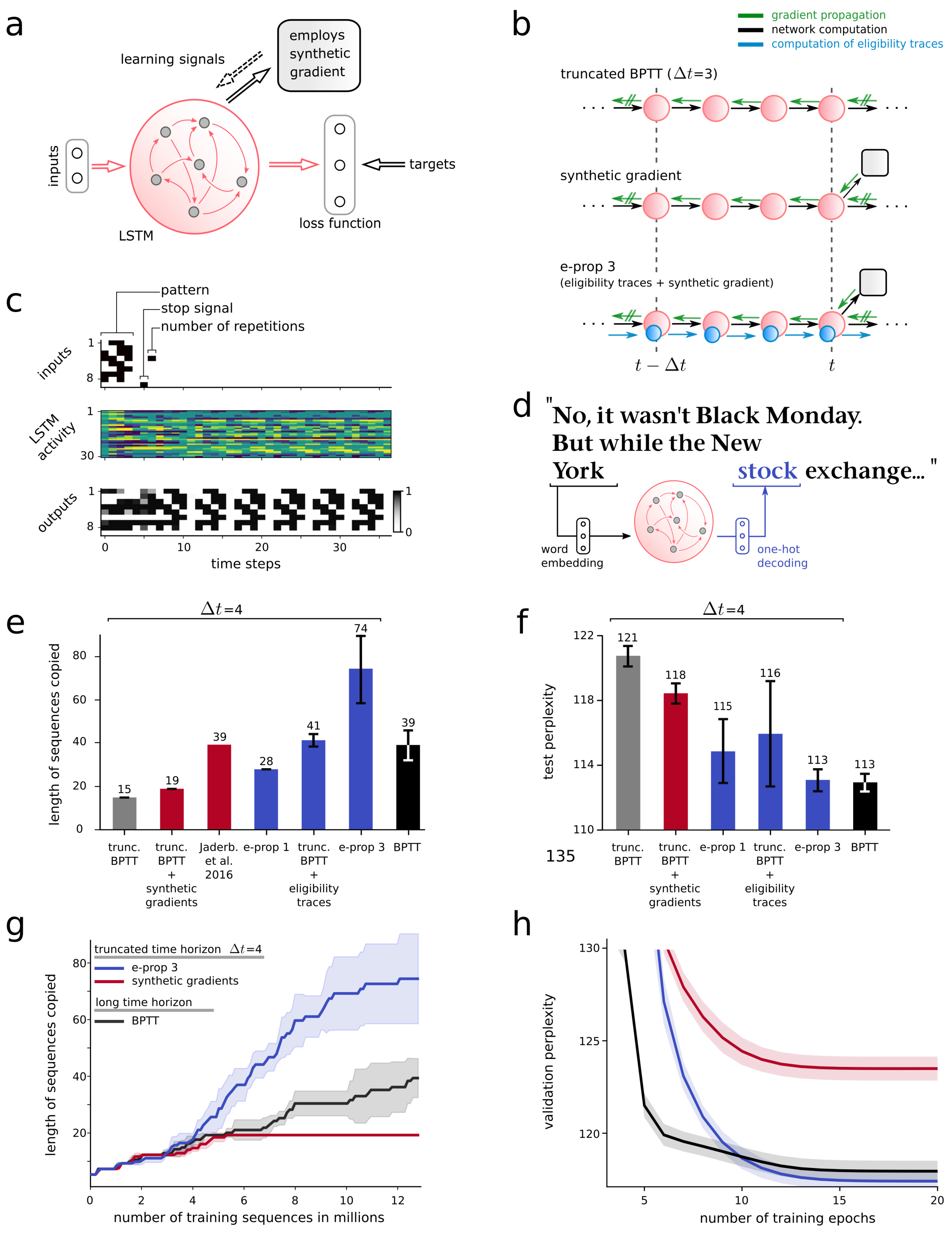


Fig. 4.4: Scheme and performance of *e-prop 3* **a**) Learning architecture for *e-prop 3*. The error module is implemented through synthetic gradients. **b**) Scheme of learning rules used in panels **e-h**. Δt is the number of time steps through which the gradients are allowed to flow for truncated BPTT or for the computation of synthetic gradients. **c**,**e**,**g**) Copy-repeat task: (**c**) example trial, (**e**) performance of different algorithms for the copy-repeat task, (**g**) Learning progress for 3 different algorithms. This task was used as a benchmark in (Jaderberg et al., 2016) and (Graves et al., 2014). **d**,**f**,**h**) Word prediction task: (**d**) example of sequence, (**f**) performance summary for different learning rules, (**h**) learning progression. An epoch denotes a single pass through the complete dataset.

4.2.3 *E-prop 3*: Producing learning signals through synthetic gradients

We show here that the use of biologically inspired eligibility traces also improves some state-of-the-art algorithms in machine learning, specifically the synthetic gradient approach for learning in feedforward and recurrent (artificial) neural networks (Jaderberg et al., 2016) and (Czarnecki et al., 2017). Synthetic gradients provide variants of backprop and BPTT that tend to run more efficiently because they avoid that synaptic weights can only be updated after a full network simulation followed by a full backpropagation of error gradients („locking"). We show here that the performance of synthetic gradient methods for recurrent neural networks significantly increases when they are combined with eligibility traces. The combination of these two approaches is in a sense quite natural, since synthetic gradients can be seen as online learning signals for *e-prop*. In comparison with *e-prop 2*, one does not need to consider here a whole family of learning tasks for L2L and a computationally intensive outer loop. Hence the production of learning signals via synthetic gradient approaches tends to be computationally more efficient. So far it also yields better results in applications to difficult tasks for recurrent artificial neural networks. In principle it is conceivable that biological learning systems also follow a strategy whereby learning signals for different temporal phases of a learning process are aligned among each other through some separate process. This is the idea of synthetic gradients.

Definition of *e-prop 3*: When BPTT is used for tasks that involve long time series, the algorithm is often truncated to shorter intervals of length Δt, and the parameters are updated after the processing of each interval. This variant of BPTT is called truncated BPTT. We show a schematic of the computation performed on the interval $\{t-\Delta t, \dots, t\}$ in the first panel of Figure 4.4b. Reducing the length Δt of the interval has two benefits: firstly, the parameter updates are more frequent; and secondly, it requires less memory storage because all inputs $\boldsymbol{x}^{t'}$ and network states $\boldsymbol{s}_j^{t'}$ for $t' \in \{t-\Delta t, \dots, t\}$ need to be stored temporarily for back-propagating gradients. The drawback is that the error gradients become more approximative, in particular, the relationship between error and network computation happening outside of this interval cannot be captured by the truncated error gradients. As illustrated in the last panel of Figure 4.4b, *e-prop 3* uses the same truncation scheme but alleviates the drawback of truncated BPTT in two ways: it uses eligibility traces to include information about the network history before $t-\Delta t$, and an error module that predicts errors after t. Thanks to eligibility traces, all input and recurrent activity patterns that happened before $t-\Delta t$ have left a footprint that can be combined with the learning signals $L_j^{t'}$ with $t' \in \{t-\Delta t, \dots, t\}$. Each of these learning signals summarizes the neuron influence on the next errors, but due to the truncation, errors performed outside of the current interval cannot trivially be taken into account. In *e-prop 3*, the error module computes learning signals $L_j^{t'}$ that anticipate the predictable components of the future errors.

E-prop 3 combines eligibility traces and learning signals to compute error gradients according to equation (4.1), rather than equation (D.3) that is canonically used to compute gradients for BPTT. To compute these gradients when processing the interval $\{t - \Delta t, \ldots, t\}$, the eligibility traces and learning signals are computed solely from data available within that interval, without requiring the rest of the data. The eligibility traces are computed in the forward direction according to equations (4.2) and (4.3). For the learning signals $L_j^{t'}$, the difficulty is to estimate the gradients $\frac{dE}{dz_j^{t'}}$ for t' between $t - \Delta t$ and t. These gradients are computed by back-propagation from $t' = t$ back to $t' = t - \Delta t + 1$ with the two recursive formulas:

$$\frac{dE}{d\boldsymbol{s}_j^{t'}} = \frac{dE}{dz_j^{t'}} \frac{\partial z_j^{t'}}{\partial \boldsymbol{s}_j^{t'}} + \frac{dE}{d\boldsymbol{s}_j^{t'+1}} \frac{\partial \boldsymbol{s}_j^{t'+1}}{\partial \boldsymbol{s}_j^{t'}} \tag{4.8}$$

$$\frac{dE}{dz_j^{t'}} = \frac{\partial E}{\partial z_j^{t'}} + \sum_i \frac{dE}{d\boldsymbol{s}_i^{t'+1}} \frac{\partial \boldsymbol{s}_i^{t'+1}}{\partial z_j^{t'}}\,, \tag{4.9}$$

which are derived by application of the chain rule at the nodes $\boldsymbol{s}_j^t$ and z_j^t of the computational graph represented in Figures D.1 ($\frac{\partial \boldsymbol{s}_i^{t+1}}{\partial z_j^t}$ is a notation short-cut for $\frac{\partial M}{\partial z_j^t}(\boldsymbol{s}_i^t, \boldsymbol{z}^t, \boldsymbol{x}^t, \boldsymbol{\theta})$, and i range over all neurons to which neuron j is synaptically connected). This leaves open the choice of the boundary condition $\frac{dE}{d\boldsymbol{s}_j^{t+1}}$ that initiates the back-propagation of these gradients at the end of the interval $\{t - \Delta t, \ldots, t\}$. In most implementations of truncated BPTT, one chooses $\frac{dE}{d\boldsymbol{s}_j^{t+1}} = 0$, as if the simulation would terminate after time t. We use instead a feed-forward neural network SG parametrized by Ψ that outputs a boundary condition SG_j associated with each neuron j: $\frac{dE}{d\boldsymbol{s}_j^{t+1}} = \mathrm{SG}_j(\boldsymbol{z}^t, \Psi)$. This strategy was proposed by (Jaderberg et al., 2016) under the name "synthetic gradients". The synthetic gradients are combined with the error gradients computed within the interval $\{t - \Delta t, \ldots, t\}$ to define the learning signals. Hence, we also see SG as a key component of the an error module in analogy with those of *e-prop 1* and 2.

To train SG, back-ups of the gradients $\frac{dE}{d\boldsymbol{s}_j^{t+1}}$ are estimated from the boundary conditions at the end of the next interval $\{t, \ldots, t + \Delta t\}$. In a similar way as value functions are approximated in reinforcement learning, these more informed gradient estimates are used as targets to improve the synthetic gradients $\mathrm{SG}(\boldsymbol{z}^t, \Psi)$. This is done in *e-prop 3* by computing simultaneously the gradients $\frac{dE'}{d\boldsymbol{\theta}}$ and $\frac{dE'}{d\Psi}$ of an error function E' that combines the error function E, the boundary condition, and the mean squared error between the synthetic gradient and its targeted back-up (see Algorithm 4 in Methods for details). These gradients are approximated to update the network parameters with any variant of stochastic gradient descent.

It was already discussed that the factorization (4.1) used in *e-prop* is equivalent to BPTT, and that both compute the error gradients $\frac{dE}{d\theta_{ji}}$. In the subsection dedicated to *e-prop 3* of Methods, we formalize the algorithm when the simulation is truncated into intervals of length Δt. We then show under the assumption that the synthetic gradients are optimal, i.e. $\mathrm{SG}_j(\boldsymbol{z}^t, \Psi) = \frac{dE}{d\boldsymbol{s}_j^{t+1}}$, that both truncated BPTT and *e-prop 3* compute the correct error gradients $\frac{dE}{d\theta_{ji}}$. However, this assumption is rarely satisfied in simulations, firstly because the parameters Ψ may not converge instantaneously to some optimum; and even then, there could be unpredictable components of the future errors that cannot be estimated correctly. Hence, a more accurate model is to assume that the synthetic gradients $\mathrm{SG}_j(\boldsymbol{z}^t, \Psi)$ are noisy estimators of $\frac{dE}{d\boldsymbol{s}_j^{t+1}}$. Under this weaker assumption it turns out that *e-prop 3* produces estimators of the error gradients $\widehat{\frac{dE}{d\theta_{ji}}}^{\text{eprop}}$ that are better than those produced with truncated BPTT with synthetic gradients $\widehat{\frac{dE}{d\theta_{ji}}}^{\text{SG}}$. Formally, this result can be summarized as:

$$\mathbb{E}\left[\left(\frac{dE}{d\theta_{ji}} - \widehat{\frac{dE}{d\theta_{ji}}}^{\text{eprop}}\right)^2\right] \leq \mathbb{E}\left[\left(\frac{dE}{d\theta_{ji}} - \widehat{\frac{dE}{d\theta_{ji}}}^{\text{SG}}\right)^2\right], \tag{4.10}$$

where the $\mathbb{E}$ is the stochastic expectation. The proof of this result will be published in a later version of the paper. To summarize the proof, we compare in detail the terms computed with *e-prop 3* and BPTT. The derivation reveals that both algorithms compute a common term that combines partial derivatives $\frac{\partial \boldsymbol{s}^t}{\partial \theta_{ji}}$ with errors, $E(\boldsymbol{z}^{t'})$ with t and t' being accessible within the interval $\{t - \Delta t, \dots, t\}$. The difference between the two algorithms is that truncated BPTT combines all the partial derivatives $\frac{\partial \boldsymbol{s}^t}{\partial \theta_{ji}}$ with synthetic gradients that predict future errors. Instead, the *e-prop* algorithm holds these partial derivatives in eligibility traces to combine them later with a better informed learning signals that do not suffer from the noisy approximations of the synthetic gradients.

Copy-repeat task 3.1 The copy-repeat task was introduced in (Graves et al., 2014) to measure how well artificial neural networks can learn to memorize and process complex patterns. It was also used in (Jaderberg et al., 2016) to compare learning algorithms for recurrent neural networks with truncated error propagation.

Task: The task is illustrated in Figure 4.4c. It requires to read a sequence of 8-bit characters followed by a "stop" character and a character that encodes the requested number of repetitions. In the subsequent time steps the network is trained to repeat the given pattern as many times as requested, followed by a final "stop" character. We used the same curriculum of increased task complexity as defined in (Jaderberg et al., 2016), where the authors benchmarked variants of truncated BPTT and synthetic gradients: the pattern length and the number of repetitions increase alternatively each time the network solves the task (the task is considered solved when the average error is below 0.15 bits per sequence). The performance of the learning algorithms are therefore measured by the length of the largest sequence for which the network could solve the task.

Implementation: The recurrent network consisted of 256 LSTM units. The component SG of the error module was a feedfoward network with one hidden layer of 512 rectified linear units (the output layer of the synthetic gradient did not have non-linear activation functions). The network states were reset to zero at the beginning of each sequence and the mean error and the error gradients were averaged over a batch of 256 independent sequences. Importantly, we used for all training algorithms a fixed truncation length $\Delta t = 4$, with the exception of a strong baseline BPTT for which the gradients were not truncated.

Results: The activity and output of a trained LSTM solving the task is displayed in Figure 4.4c. The performance of various learning algorithms is summarized in Figure 4.4e and g. Truncated BPTT alone solves the task for sequences of 15 characters, and 19 characters when enhanced with synthetic gradients. With a different implementation of the task and algorithm, (Jaderberg et al., 2016) reported that sequences of 39 characters could be handled with synthetic gradients. When BPTT is replaced by *e-prop* and uses eligibility traces, the network learnt to solve the task for sequences of length 41 when the synthetic gradients were set to zero (this algorithm is referred as "truncated BPTT + eligibility traces" in Figure 4.4e and f). The full *e-prop 3* algorithm that includes eligibility traces and synthetic gradients solved the task for sequences of 74 characters. This is an improvement over *e-prop 1* that handles sequences of 28 characters, even if the readout weights are not replaced by random error broadcasts. All these results were achieved with a truncation length $\Delta t = 4$. In contrast when applying full back-propagation through the whole sequence, we reached only 39 characters.

Word prediction task 3.2 We also considered a word-level prediction task in a corpus of articles extracted from the Wall Street Journal, provided by the so-called Penn Treebank dataset. As opposed to the previous copy-repeat task, this is a practically relevant task. It has been standardized to provide a reproducible benchmark task. Here, we used the same implementation and baseline performance as provided freely by the Tensorflow tutorial on recurrent neural networks[1].

Task: As indicated in Figure 4.4d, the network reads the whole corpus word after word. At each time step, the network has to read a word and predict the following one. The training, validation and test sets consist of texts of 929k, 73k, and 82k words. The sentences are kept in a logical order such that the context of dozens of words matters to accurately predict the following ones. The vocabulary of the dataset is restricted to the 10k most frequent words, and the words outside of this vocabulary are replaced with a special unknown word token.

Implementation: For all algorithms, the parameters were kept identical to those defined in the Tensorflow tutorial, with two exceptions: firstly, the networks had a single layer of 200 LSTM units instead of two to simplify the implementation, and because the second did not seem to improve performance with this configuration;

[1]https://www.tensorflow.org/tutorials/sequences/recurrent

secondly, the truncation length was reduced from $\Delta t = 20$ to $\Delta t = 4$ for synthetic gradients and *e-prop 3* to measure how our algorithms compensate for it. The synthetic gradients are computed by a one hidden layer of 400 rectified linear units. For a detailed description of model and a list of parameters we refer to the methods.

Results: The error is measured for this task by the word-level perplexity, which is the exponential of the mean cross-entropy loss. Figure 4.4e and f summarize our results. After reduction to a single layer, the baseline perplexity of the model provided in the Tensorflow tutorial for BPTT was 113 with a truncation length $\Delta t = 20$. Full BPTT is not practical in this case because the data consists of one extremely long sequence of words. In contrast, in a perplexity increased to 121 when the same model was also trained with BPTT, but a shorter truncation length $\Delta t = 4$. The model performance improved back to 118 with synthetic gradients, and to 116 with eligibility traces. Applying the *e-prop 1* algorithm with the true readout weights as error broadcasts resulted in a perplexity of 115. When combining both eligibility traces and synthetic gradients in *e-prop 3*, the performance improved further and we achieved a perplexity of 113.

To further investigate the relevance of eligibility traces for *e-prop 3* we considered the case where the eligibility trace were truncated. Instead of using the eligibility trace vectors as defined in equation (D.27) we used $\boldsymbol{\epsilon}_{ji}^t = \frac{\partial \boldsymbol{s}_j^t}{\partial \theta_{ji}}$. This variation of *e-prop 3* lead to significantly degraded performance and resulted in test perplexity of 122.67 (not shown).

All together Figure 4.4e and f show that eligibility traces improve truncated BPTT more than synthetic gradients. Furthermore, if eligibility traces are combined with synthetic gradients in *e-prop 3*, one arrives at an algorithm that outperforms full BPTT for the copy-repeat task, and matches the performance of BPTT ($\Delta t = 20$) for the Penn Treebank word prediction task.

4.3 Discussion

The functionally most powerful learning method for recurrent neural nets, an approximation of gradient descent for a loss function via BPTT, requires propagation of error signals backwards in time. Hence this method does not reveal how recurrent networks of neurons in the brain learn. In addition, propagation of error signals backwards in time requires costly work-arounds in software implementations, and it does not provide an attractive blueprint for the design of learning algorithms in neuromorphic hardware. We have shown that a replacement of the propagation of error signals backwards in time in favor of a propagation of the local activation histories of synapses – called eligibility traces – forward in time allows us to capture with physically and biologically realistic mechanisms a large portion of the functional benefits of BPTT. We are referring to to this new approach to gradient descent learning in recurrent neural networks as *e-prop*. In contrast to many other approaches for learning in recurrent networks of spiking neurons it can be based on a rigorous mathematical theory.

We have presented a few variations of *e-prop* where eligibility traces are combined with different types of top-down learning signals that are generated and transmitted in real-time. In *e-prop 1* we combine eligibility traces with a variation of broadcast alignment (Samadi et al., 2017) or direct feedback alignment (Nøkland, 2016). We first evaluated the performance of *e-prop 1* on a task that has become a standard for the evaluation of the FORCE learning method for recurrent networks of spiking neurons (Nicola and Clopath, 2017) and related earlier work on artificial neural networks: supervised learning of generating a temporal pattern. In order to make the task more interesting we considered a task where 3 independent temporal patterns have to be generated simultaneously by the same RSNN. Here it is not enough to transmit a single error variable to the network, so that broadcasting of errors for different dimensions of the network output to the network becomes less trivial. We found (see Figure 4.1) that a random weight matrix for the distribution of error signals works well, as in the case of feedforward networks (Samadi et al., 2017), (Nøkland, 2016). But surprisingly, the results were best when this matrix was fixed, or rarely changed, whereas a direct application of broadcast alignment to an unrolled recurrent network suggests that a different random matrix should be used for every time slice.

In order to challenge the capability of *e-prop 1* to deal also with cases where error signals arise only at the very end of a computation in a recurrent network, we considered a store-recall task, where the network has to learn what information it should store –and maintain until it is needed later. We found (see Figure 4.2) that this task can also be learnt with *e-prop 1*, and verified that the eligibility traces of the network were able to bridge the delay. We used here an LSNN (Bellec et al., 2018c) that includes a model of a slower process in biological neurons: neuronal adaptation. We also compared the performance of *e-prop 1* with BPTT for a more demanding task: the speech recognition benchmark task TIMIT. We found that *e-prop 1* approximates also here the performance of BPTT quite well.

Altogether we have the impression that *e-prop 1* can solve all learning tasks for RSNNs that the FORCE method can solve, and many more demanding tasks. Since the FORCE method is not argued to be biologically realistic, whereas *e-prop 1* only relies on biologically realistic mechanisms, this throws new light on the understanding of learning in recurrent networks of neurons in the brain. An additional new twist is that *e-prop 1* engages also plasticity of synaptic connections within a recurrent network, rather than only synaptic connections to a postulated readout neuron as in the FORCE method. Hence we can now analyze how network configurations and motifs that emerge in recurrent neural network models through learning (possibly including in e-prop biologically inspired rewiring mechanisms as in (Bellec et al., 2018a)) relate to experimental data.

In the analysis of *e-prop 2* we moved to a minimal model that captures salient aspects of the organization of learning in the brain, where dedicated brain areas process error signals and generate suitably modified gating signals for plasticity in different populations of neurons. We considered in this minimal model just a single RSNN (termed error module) for generating learning signals. But obviously this minimal model opens the door to the analysis of more complex learning architectures –as they are found in the brain– from a functional perspective. We found that a straightforward application of the Learning-to-Learn (L2L) paradigm, where the error module is optimized on a slower time scale for its task, significantly boosts the learning capability of an RSNN. Concretely, we found that it endows the RSNN with one-shot learning capability (see Figure 4.3), hence with a characteristic advantage of learning in the human brain (Brea and Gerstner, 2016), (Lake et al., 2017). In addition the model of Figure 4.3 suggests a new way of thinking about motor learning. It is shown that no separate inverse model is needed to learn motor control. Furthermore in the case that we considered, not even sensory feedback from the environment is needed.

Finally we arrived at an example where biologically inspired ideas and mechanisms, in this case eligibility traces, can enhance state-of-the-art methods in machine learning. Concretely, we have shown in Figure 4.4 that adding eligibility traces to the synthetic gradient methods of (Jaderberg et al., 2016) and (Czarnecki et al., 2017) for training artificial recurrent neural networks significantly enhances the performance of synthetic gradient algorithms. In fact, the resulting algorithm *e-prop 3* was found to supercede the performance of full BPTT in one case and rival BPTT with $\Delta t = 20$ in another.

A remarkable feature of *e-prop* is that the resulting local learning rules (D.14) are very similar to previously proposed rules for synaptic plasticity that were fitted to experimental data (Clopath et al., 2010). In fact, we have shown in Figure 4.1d that the theory-derived local plasticity rule for *e-prop 1* can be replaced by the Clopath rule with little loss in performance. On a more general level, the importance of eligibility traces for network learning that our results suggest provides concrete hypotheses for the functional role of a multitude of processes on the molecular level in neurons and synapses, including metabotropic receptors. Many of these processes are known to store information about multiple aspects of the recent history. The *e-prop* approach suggests that these processes, in combination with a sufficiently sophisticated production of learning signals by dedicated brain structures, can practically replace the physically impossible backpropagation of error signals backwards in time of theoretically optimal BPTT.

An essential prediction of e-prop for synaptic plasticity rules is that learning signals can switch the sign of synaptic plasticity, i.e., between LTP and LTD or between STDP and anti-STDP. Such switching of the sign of plasticity via disinhibition had been found in synapses from the cortex to the striatum (Paille et al., 2013), see (Perrin and Venance, 2019) for a recent review. Further brain mechanisms for switching the sign of plasticity through 3rd factors had been reported in (C. H. Chen et al., 2014; Cui et al., 2016; Foncelle et al., 2018).

A key challenge for neuromorphic engineering is to design a new generation of computing hardware that enables energy efficient implementations of major types of networks and learning algorithms that have driven recent progress in machine learning and learning-driven AI (see e.g. (Barrett et al., 2018)). Recurrent neural networks are an essential component of many of these networks, and hence learning algorithms are needed for this type of networks that can be efficiently implemented in neuromorphic hardware. In addition, neuromorphic implementations of recurrent neural networks – rather than deep feedforward networks—promise larger efficiency gains because hardware neurons can be re-used during a computation. Recently developed diffusive memristors (Z. Wang et al., 2018) would facilitate an efficient local computation of eligibility traces with new materials. In addition, new 3-terminal memristive synapses (Y. Yang et al., 2017) are likely to support an efficient combination of local eligibility traces with top-down error signals in the hardware. Thus *e-prop* provides attractive functional goals for novel materials in neuromorphic hardware.

Appendix

Appendix A

List of publications

1. 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).*
2. Guillaume BEllec⋆, Franz Scherr⋆, Elias Hajek, Darjan Salaj, Robert Legenstein, Wolfgang Maass (2019). "Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets." *arxiv.*
3. Chen Liu⋆, Guillaume BEllec⋆, Bernhard Vogginger, David Kappel, Johannes Partzsch, Felix Neumärker, Sebastian Höppner, Wolfgang Maass, Steve B Furber, Robert Legenstein, Christian G Mayr (2018). "Memory-efficient deep learning on a SpiNNaker 2 prototype." *Frontiers in neuroscience | Neuromorphic engineering.*
4. 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 (NeurIPS).*
5. Guillaume BEllec, David Kappel, Wolfgang Maass, Robert Legenstein (2018). "Deep rewiring: Training very sparse deep networks." *International Conference on Machine Learning (ICLR).*
6. 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 systems." *IEEE International Symposium on Circuits and Systems (ISCAS).*
7. Sebastian Schmitt, Johann Klähn, Guillaume BEllec, Andreas Grübl, Maurice Guettler, 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, Mihai A Petrovici, Stefan Schiefer, Stefan Scholze, Vasilis Thanasoulis, Bernhard Vogginger, Robert Legenstein, Wolfgang Maass, Christian Mayr, René Schüffny, Johannes Schemmel, Karlheinz Meier (2017). "Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system." *International Joint Conference on Neural Networks (IJCNN).*

8. Guillaume BEllec, Mathieu Galtier, Romain Brette, Pierre Yger (2016). "Slow feature analysis with spiking neurons and its application to audio stimuli." *Journal of computational neuroscience.*

⋆ equal contribution

Appendix B

Appendix to “Deep Rewiring: Training very sparse deep networks”

B.1 Methods

Implementations of DEEP R are freely available at github.com/guillaumeBellec/deep_rewiring.

Choosing hyper-parameters for DEEP R: The learning rate η is defined for each task independently (see task descriptions below). Considering that the number of active connections is given as a constraint, the remaining hyper parameters are the regularization coefficient α and the temperature T. We found that the performance of DEEP R does not depend strongly on the temperature T. Yet, the choice of α has to be done more carefully. For each dataset there was an ideal value of α: one order of magnitude higher or lower typically lead to a substantial loss of accuracy.

In MNIST, 96.3% accuracy under the constraint of 1% connectivity was achieved with $\alpha = 10^{-4}$ and T chosen so that $T = \frac{\eta}{2}10^{-12}$. In TIMIT, $\alpha = 0.03$ and $T = 0$ (higher values of T could improve the performance slightly but it did not seem very significant). In CIFAR-10 a different α was assigned to each connectivity matrix. To reach 84.1% accuracy with 5% connectivity we used in each layer from input to output $\alpha = [0, 10^{-7}, 10^{-6}, 10^{-9}, 0]$. The temperature is initialized with $T = \eta\frac{\alpha^2}{18}$ and decays with the learning rate (see paragraph of the methods about CIFAR-10).

Choosing hyper-parameters for soft-DEEP R: The main difference between soft-DEEP R and DEEP R is that the connectivity is not given as a global constraint. This is a considerable drawback if one has strict constraint due to hardware limitation but it is also an advantage if one simply wants to generate very sparse network solutions without having a clear idea on the connectivities that are reachable for the task and architecture considered.

In any cases, the performance depends on the choice of hyper-parameters α, T and θ_{min}, but also - unlike in DEEP R - these hyper parameters have inter-dependent relationships that one cannot ignore (as for DEEP R, the learning rate η is defined for each task independently). The reason why soft-DEEP R depends more on the temperature is that the rate of re-activation of connections is driven by the amplitude of the noise whereas they are decoupled in DEEP R. To summarize the results of an exhaustive parameter search, we found that $\sqrt{2T\eta}$ should ideally be slightly below α. In general high θ_{min} leads to high performance but it also defines an approximate lower bound on the smallest reachable connectivity. This lower bound can be estimated by computing analytically the stationary distribution under rough approximations and the assumption that the gradient of the likelihood is zero. If p_{min} is the targeted lower connectivity bound, one needs $\theta_{min} \approx -\frac{T(1-p_{min})}{\alpha p_{min}}$.

For MNIST we used $\alpha = 10^{-5}$ and $T = \eta \frac{\alpha^2}{18}$ for all data points in Fig. 2.1 panel A and a range of values of θ_{min} to scope across different ranges of connectivity lower bounds. In TIMIT and CIFAR-10 we used a simpler strategy which lead to a similar outcome, we fixed the relationships: $\alpha = 3\sqrt{2\frac{T}{\eta}} = \frac{-1}{3}\theta_{min}$ and we varied only α to produce the solutions shown in Fig. 2.1 panel B and Fig. 2.2.

Re-implementing pruning and ℓ_1-shrinkage: To implement ℓ_1-shrinkage (Tibshirani, 1996; Collins and Kohli, 2014), we applied the ℓ_1-shrinkage operator $\theta \leftarrow \text{relu}\left(|\theta| - \eta\alpha\right)\text{sign}(\theta)$ after each gradient descent iteration. The performance of the algorithm is evaluated for different α varying on a logarithmic scale to privilege a sparse connectivity or a high accuracy. For instance for MNIST in Figure 2.3.A we used α of the form $10^{-\frac{n}{2}}$ with n going from 4 to 12. The optimal parameter was $n = 9$.

We implemented the pruning described in Han et al., 2015b. This algorithm uses several phases: training - pruning - training, or one can also add another pruning iteration: training - pruning - training - pruning - training. We went for the latter because it increased performance. Each "training" phase is a complete training of the neural network with ℓ_2-regularization[1]. At each "pruning" phase, the standard deviation of weights within a weight matrix w_{std} is computed and all active weights with absolute values smaller than qw_{std} are pruned (q is called the quality parameter). Grid search is used to optimize the ℓ_2-regularization coefficient and quality parameter. The results for MNIST are reported in Figure B.1.

MNIST: We used a standard feed forward network architecture with two hidden layers with 200 neurons each and rectified linear activation functions followed by a 10-fold softmax output. For all algorithms we used a learning rate of 0.05 and a batch size of 10 with standard stochastic gradient descent. Learning stopped after 10 epochs. All reported performances in this article are based on the classification error on the MNIST test set.

[1]To be fair with other algorithms, we did not allocate three times more training time to pruning, each "training" phase was performed for a third of the total number of epochs which was chosen much larger than necessary.

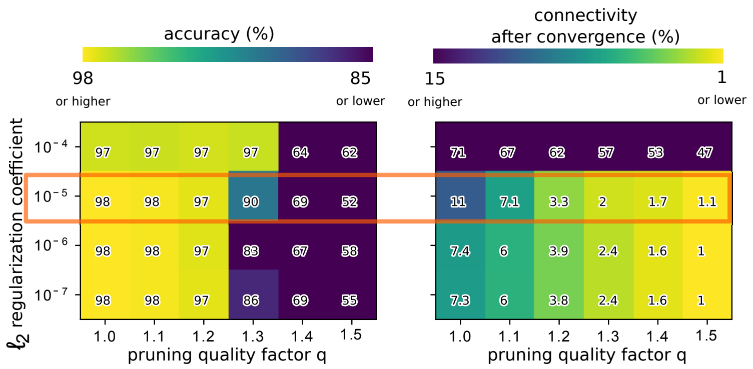


Fig. B.1: Hyper-parameter search for the pruning algorithm according to Han et al., 2015b. Each point of the grid represents a weight decay coefficient – quality factor pair. The number and the color indicate the performance in terms of accuracy (left) or connectivity (right). The red rectangle indicates the data points that were used in Fig. 2.3A.

CIFAR-10: The official tutorial for convolutional networks of tensorflow[2] is used as a reference implementation. Its performance out-of-the-box provides the fully connected baseline. We used the values given in the tutorial for the hyper-parameters in all algorithms. In particular the layer-specific weight decay coefficients that interact with our algorithms were chosen from the tutorial for DEEP R, soft-DEEP R, pruning, and ℓ_1-shrinkage.

In the fully connected baseline implementation, standard stochastic gradient descent was used with a decreasing learning rate initialized to 1 and decayed by a factor 0.1 every 350 epochs. Training was performed for one million iterations for all algorithms. For soft-DEEP R, which includes a temperature parameter, keeping a high temperature as the weight decays was increasing the rate of re-activation of connections. Even if intermediate solutions were rather sparse and efficient the solutions after convergence were always dense. Therefore, the weight decay was accompanied by annealing of the temperature T. This was done by setting the temperature to be proportional to the decaying η. This annealing was used for DEEP R and soft-DEEP R.

TIMIT: The TIMIT dataset was preprocessed and the LSTM architecture was chosen to reproduce the results from Greff et al., 2017. Input time series were formed by 12 MFCC coefficients and the log energy computed over each time frame. The inputs were then expanded with their first and second temporal derivatives. There are 61 different phonemes annotated in the TIMIT dataset, to report an error rate that is comparable to the literature we performed a standard grouping of the phonemes to generate 39 output classes (Lee and Hon, 1989; Graves et al., 2013; Greff et al., 2017). As usual, the dialect specific sentences were excluded (SA files). The phoneme error rate was computed as the proportion of misclassified frames.

[2]TensorFlow version 1.3: www.tensorflow.org/tutorials/deep_cnn

A validation set and early stopping were necessary to train a network with dense connectivity matrix on TIMIT because the performance was sometimes unstable and it suddenly dropped during training as seen in Fig. 2.3D for ℓ_1-shrinkage. Therefore a validation set was defined by randomly selecting 5% of the training utterances. All algorithms were trained for 40 epochs and the reported test error rate is the one at minimal validation error.

To accelerate the training in comparison the reference from Greff et al., 2017 we used mini-batches of size 32 and the ADAM optimizer (Kingma and Ba, 2014). This was also an opportunity to test the performance of DEEP R and soft-DEEP R with such a variant of gradient descent. The learning rate was set to 0.01 and we kept the default momentum parameters of ADAM, yet we found that changing the ϵ parameter (as defined in Kingma and Ba, 2014) from 10^{-8} to 10^{-4} improved the stability of fully connected networks during training in this recurrent setup. As we could not find a reference that implemented ℓ_1-shrinkage in combination with ADAM, we simply applied the shrinkage operator after each iteration of ADAM which might not be the ideal choice in theory. It worked well in practice as the minimal error rate was reached with this setup. The same type of ℓ_1 regularization in combination with ADAM was used for DEEP R and soft-DEEP R which lead to very sparse and efficient network solutions.

Initialization of connectivity matrices: We found that the performance of the networks depended strongly on the initial connectivity. Therefore, we followed the following heuristics to generate initial connectivity for DEEP R, soft-DEEP R and the control setup with fixed connectivity.

First, for the connectivity matrix of each individual layer, the zero entries were chosen with uniform probability. Second, for a given connectivity constraint we found that the learning time increased and the performance dropped if the initial connectivity matrices were not chosen carefully. Typically the performance dropped drastically if the output layer was initialized to be very sparse. Yet in most networks the number of parameters is dominated by large connectivity matrices to hidden layers. A basic rule of thumb that worked in our cases was to give an equal number of active connections to the large and intermediate weight matrices, whereas smaller ones - typically output layers - should be densely connected.

We suggest two approaches to refine this guess: One can either look at the statistics of the connectivity matrices after convergence of DEEP R or soft-DEEP R, or, if possible, the second alternative is to initialize once soft-DEEP R with a dense matrix and observe the connectivity matrix after convergence. In our experiments the connectivities after convergence were coherent with the rule of thumb described above and we did not need to pursue intensive search for ideal initial connectivity matrices.

For MNIST, the number of parameters in each layer was 235k, 30k and 1k from input to output. Using our rule of thumb, for a given global connectivity p_0, the layers were respectively initialized with connectivity $0.75p_0$, $2.3p_0$ and $22.8p_0$.

For CIFAR-10, the baseline network had two convolutional layers with filters of shapes $5 \times 5 \times 3 \times 64$ and $5 \times 5 \times 64 \times 64$ respectively, followed by two fully connected layer with weight matrices of shape 2304×384 and 384×192. The last layer was then projected into a softmax over 10 output classes. The numbers of parameters per connectivity matrices were therefore 5k, 102k, 885k, 74k and 2k from input to output. The connectivity matrices were initialized with connectivity $1, 4p_0, 0.4p_0, 4p_0$, and 1 where p_0 is approximately the resulting global connectivity.

For TIMIT, the connection matrix from the input to the hidden layer was of size 39×800, the recurrent matrix had size 200×800 and the size of the output matrix was 200×39. Each of these three connectivity matrices were respectively initialized with a connectivity of $1.8p_0, 0.6p_0$, and $6p_0$ where p_0 is approximately the resulting global connectivity.

Initialization of weight matrices: For CIFAR-10 the initialization of matrix coefficients was given by the reference implementation. For MNIST and TIMIT, the weight matrices were initialized with $\boldsymbol{\theta} = \frac{1}{\sqrt{n_{in}}}\mathcal{N}(0,1)\mathbf{c}$ where n_{in} is the number of afferent neurons, $\mathcal{N}(0,1)$ samples from a centered gaussian with unit variance and $\mathbf{c}$ is a binary connectivity matrix.

It would not be good to initialize the parameters of all dormant connections to zero in soft-DEEP R. After a single noisy iteration, half of them would become active which would fail to initialize the network with a sparse connectivity matrix. To balance out this problem we initialized the parameters of dormant connections uniformly between the clipping value θ_{min} and zero in soft-DEEP R.

Parameters for Figure 2.4 The experiment provided in Figure 2.4 is a variant of our MNIST experiment where the target labels were shuffled after every training epoch. To make the generalization capability of DEEP R over a small number of epochs visible, we enhanced the noise exploration by setting a batch to 1 so that the connectivity matrices were updated at every time step. Also we used a larger network with 400 neurons in each hidden layer. The remaining parameters were similar to those used previously: the connectivity was constrained to 1% and the connectivity matrices were initialized with respective connectivities: 0.01, 0.01, and 0.1. The parameters of DEEP R were set to $\eta = 0.05$, $\alpha = 10^{-5}$ and $T = \eta\frac{\alpha^2}{2}$.

B.2 Rewiring during training on MNIST

Fig. B.2 shows the rewiring behavior of DEEP R per network layer for the feed-forward neural network trained on MNIST and the training run indicated by the

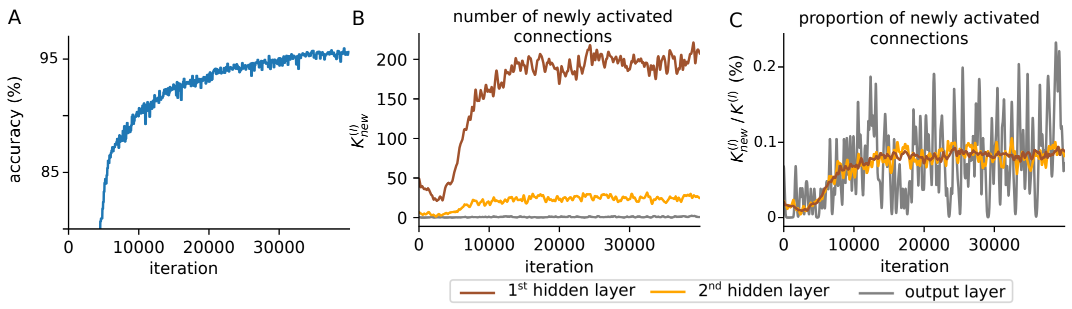


Fig. B.2: Rewiring behavior of DEEP R. A) Network performance versus training iteration (same as green line in Fig. 2.1A bottom, but for a network constrained to 1% connectivity). **B)** Absolute number of newly activated connections $K^{(l)}_{\text{new}}$ to layer $l = 1$ (brown), $l = 2$ (orange), and the output layer ($l = 3$, gray) per iteration. Note that these layers have quite different numbers of potential connections $K^{(l)}$. **C)** Same as panel B but the number of newly activated connections are shown relative to the number of potential connections in the layer (values in panel C are smoothed with a boxcar filter over X iterations).

small gray box around the green dot in Fig. 2.1A. Since it takes some iterations until the weights of connections that do not contribute to a reduction of the error are driven to 0, the number of newly established connections $K^{(l)}_{\text{new}}$ in layer l is small for all layers initially. After this initial transient, the number of newly activated connections stabilized to a value that is proportional to the total number of potential connections in the layer (Fig. 2.1B). DEEP R continued to rewire connections even late in the training process.

B.3 Details to: Convergence properties of soft-DEEP R

Here we provide additional details on the convergence properties of the soft-DEEP R parameter update provided in Algorithm 2. We reiterate here Eq. (2.2):

$$d\theta_k \;=\; \beta\,\frac{\partial}{\partial\theta_k}\log p^*(\boldsymbol{\theta}|\mathbf{X},\mathbf{Y}^*)\Big|_{\boldsymbol{\theta}^t} dt \;+\; \sqrt{2\beta T}\,d\mathcal{W}_k\,. \tag{B.1}$$

Discrete time updates can be recovered from the set of SDEs (B.1) by integration over a short time period Δt

$$\Delta\theta_k = \eta\frac{\partial}{\partial\theta_k}\log p^*(\boldsymbol{\theta}|\mathbf{X},\mathbf{Y}^*) + \sqrt{2\eta T}\;\nu_k, \tag{B.2}$$

where the learning rate η is given by $\eta = \beta\,\Delta t$.

We prove that the stochastic parameter dynamics Eq. (B.1) converges to the target distribution $p^*(\boldsymbol{\theta})$ given in Eq. (2.3). The proof is analogous to the derivation given in Kappel et al., 2015; Kappel et al., 2018. We reiterate the proof here for the special case of supervised learning. The fundamental property of the synaptic sampling dynamics Eq. (B.1) is formalized in Theorem 1 and proven below. Before we state the theorem, we briefly discuss its statement in simple terms. Consider some initial parameter setting $\boldsymbol{\theta}^0$. Over time, the parameters change according to the dynamics (B.1). Since the dynamics include a noise term, the exact value of the parameters $\boldsymbol{\theta}(t)$ at some time $t > 0$ cannot be determined. However, it is possible to describe the exact distribution of parameters for each time t. We denote this distribution by $p_{\mathrm{FP}}(\boldsymbol{\theta}, t)$, where the "FP" subscript stands for "Fokker-Planck" since the evolution of this distribution is described by the Fokker-Planck equation (B.3) given below. Note that we make the dependence of this distribution on time explicit in this notation. It can be shown that for the dynamics (B.3), $p_{\mathrm{FP}}(\boldsymbol{\theta}, t)$ converges to a well-defined and unique *stationary distribution* in the limit of large t. To prove the convergence to the stationary distribution we show that it is kept invariant by the set of SDEs Eq. (B.1) and that it can be reached from any initial condition.

We now state Theorem 1 formally. To simplify notation we drop in the following the explicit time dependence of the parameters $\boldsymbol{\theta}$.

Theorem 1. *Let $p^*(\boldsymbol{\theta} \,|\, \mathbf{X}, \mathbf{Y}^*)$ be a strictly positive, continuous probability distribution over parameters $\boldsymbol{\theta}$, twice continuously differentiable with respect to $\boldsymbol{\theta}$, and let $\beta > 0$. Then the set of stochastic differential equations Eq.* (B.1) *leaves the distribution $p^*(\boldsymbol{\theta})$* (2.3) *invariant. Furthermore, $p^*(\boldsymbol{\theta})$ is the unique stationary distribution of the sampling dynamics.*

Proof. The stochastic differential equation Eq. (B.1) translates into a Fokker-Planck equation (Gardiner, n.d.) that describes the evolution of the distribution over parameters $\boldsymbol{\theta}$

$$\frac{\partial}{\partial t} p_{\mathrm{FP}}(\boldsymbol{\theta}, t) = \sum_k -\frac{\partial}{\partial \theta_k}\left(\beta \frac{\partial}{\partial \theta_k} \log p^*(\boldsymbol{\theta} \,|\, \mathbf{X}, \mathbf{Y}^*)\right) p_{\mathrm{FP}}(\boldsymbol{\theta}, t) + \frac{\partial^2}{\partial \theta_k^2}\left(\beta\, T\, p_{\mathrm{FP}}(\boldsymbol{\theta}, t)\right), \tag{B.3}$$

where $p_{\mathrm{FP}}(\boldsymbol{\theta}, t)$ denotes the distribution over network parameters at time t. To show that $p^*(\boldsymbol{\theta})$ leaves the distribution invariant, we have to show that $\frac{\partial}{\partial t} p_{\mathrm{FP}}(\boldsymbol{\theta}, t) = 0$ (i.e., $p_{\mathrm{FP}}(\boldsymbol{\theta}, t)$ does not change) if we set $p_{\mathrm{FP}}(\boldsymbol{\theta}, t)$ to $p^*(\boldsymbol{\theta})$. Plugging in the presumed stationary distribution $p^*(\boldsymbol{\theta})$ for $p_{\mathrm{FP}}(\boldsymbol{\theta}, t)$ on the right hand side of Eq. (B.3), one obtains

$$\begin{aligned}
\frac{\partial}{\partial t} p_{\mathrm{FP}}(\boldsymbol{\theta}, t) &= \sum_k -\frac{\partial}{\partial \theta_k}\left(\beta \frac{\partial}{\partial \theta_k} \log p^*(\boldsymbol{\theta} \,|\, \mathbf{X}, \mathbf{Y}^*)\, p^*(\boldsymbol{\theta})\right) + \frac{\partial^2}{\partial \theta_k^2}\left(\beta\, T\, p^*(\boldsymbol{\theta})\right) \\
&= \sum_k -\frac{\partial}{\partial \theta_k}\left(\beta\, p^*(\boldsymbol{\theta}) \frac{\partial}{\partial \theta_k} \log p^*(\boldsymbol{\theta} \,|\, \mathbf{X}, \mathbf{Y}^*)\right) + \frac{\partial}{\partial \theta_k}\left(\beta\, T \frac{\partial}{\partial \theta_k} p^*(\boldsymbol{\theta})\right) \\
&= \sum_k -\frac{\partial}{\partial \theta_k}\left(\beta\, p^*(\boldsymbol{\theta}) \frac{\partial}{\partial \theta_k} \log p^*(\boldsymbol{\theta} \,|\, \mathbf{X}, \mathbf{Y}^*)\right) + \frac{\partial}{\partial \theta_k}\left(\beta\, T\, p^*(\boldsymbol{\theta}) \frac{\partial}{\partial \theta_k} \log p^*(\boldsymbol{\theta})\right),
\end{aligned}$$

which by inserting $p^*(\boldsymbol{\theta}) = \frac{1}{\mathcal{Z}} p^*(\boldsymbol{\theta} \,|\, \mathbf{X}, \mathbf{Y}^*)^{\frac{1}{T}}$, with normalizing constant $\mathcal{Z}$, becomes

$$\frac{\partial}{\partial t} p_{\mathrm{FP}}(\boldsymbol{\theta}, t) = \frac{1}{\mathcal{Z}} \sum_k -\frac{\partial}{\partial \theta_k} \left(\beta\, p^*(\boldsymbol{\theta}) \frac{\partial}{\partial \theta_k} \log p^*(\boldsymbol{\theta} \,|\, \mathbf{X}, \mathbf{Y}^*) \right) + \frac{\partial}{\partial \theta_k} \left(\beta\, T\, p^*(\boldsymbol{\theta}) \frac{1}{T} \frac{\partial}{\partial \theta_k} \log p^*(\boldsymbol{\theta} \,|\, \mathbf{X}, \mathbf{Y}^*) \right) \;=\; \sum_k 0 \;=\; 0\,.$$

This proves that $p^*(\boldsymbol{\theta})$ is a stationary distribution of the parameter sampling dynamics Eq. (B.1). Since β is positive by construction, the Markov process of the SDEs (B.1) is ergodic and the stationary distribution is unique (see Section 5.3.3. and 3.7.2 in Gardiner, n.d.).

The unique stationary distribution of Eq. (B.3) is given by $p^*(\boldsymbol{\theta}) = \frac{1}{\mathcal{Z}} p^*(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y}^*)^{\frac{1}{T}}$, i.e., $p^*(\boldsymbol{\theta})$ is the only solution for which $\frac{\partial}{\partial t} p_{\mathrm{FP}}(\boldsymbol{\theta}, t)$ becomes 0, which completes the proof. □

The updates of the soft-DEEP R algorithm (Algorithm 2) can be written as

$$\Delta\theta_k = \begin{cases} \sqrt{2T\eta}\; \nu_k & \text{if } \theta_k < 0 \text{ (dormant connection)} \\ -\eta \frac{\partial}{\partial \theta_k} E_{\mathbf{X},\mathbf{Y}^*}(\boldsymbol{\theta}) - \eta\alpha + \sqrt{2T\eta}\; \nu_k & \text{otherwise.} \end{cases} \tag{B.4}$$

Eq. (B.4) is a special case of the general discrete parameter dynamics (B.2). To see this we apply Bayes' rule to expand the derivative of the log posterior into the sum of the derivatives of the prior and the likelihood:

$$\frac{\partial}{\partial \theta_k} \log p^*(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y}^*) \;=\; \frac{\partial}{\partial \theta_k} \log p_{\mathcal{S}}(\boldsymbol{\theta}) + \frac{\partial}{\partial \theta_k} \log p_{\mathcal{N}}(\mathbf{Y}^* \,|\, \mathbf{X}, \boldsymbol{\theta}) \;,$$

such that we can rewrite Eq. (B.2)

$$\Delta\theta_k = \eta \left(\frac{\partial}{\partial \theta_k} \log p_{\mathcal{S}}(\boldsymbol{\theta}) + \frac{\partial}{\partial \theta_k} \log p_{\mathcal{N}}(\mathbf{Y}^* \,|\, \mathbf{X}, \boldsymbol{\theta}) \right) + \sqrt{2\eta T}\; \nu_k, \tag{B.5}$$

To include automatic network rewiring in our deep learning model we adopt the approach described in Kappel et al., 2015. Instead of using the network parameters $\boldsymbol{\theta}$ directly to determine the synaptic weights of network $\mathcal{N}$, we apply a nonlinear transformation $w_k = f(\theta_k)$ to each connection k, given by the function

$$w_k \;=\; f(\theta_k) \;=\; s_k \frac{1}{\gamma} \log\left(1 + \exp(\gamma\, s_k\, \theta_k)\right) \;, \tag{B.6}$$

where $s_k \in \{1, -1\}$ is a parameter that determines the sign of the connection weight and $\gamma > 0$ is a constant parameter that determines the smoothness of the mapping. In the limit of large γ Eq. (B.6) converges to the rectified linear function

$$w_k \;=\; f(\theta_k) \;=\; \begin{cases} 0 & \text{if } \theta_k < 0 \quad (\textit{dormant connection}) \\ s_k\, \theta_k & \text{else} \quad (\textit{active connection}) \end{cases} \;, \tag{B.7}$$

such that all connections with $\theta_k < 0$ are not functional.

Using this, the gradient of the log-likelihood function $\frac{\partial}{\partial\theta_k} \log p_{\mathcal{N}}\left(\mathbf{Y}^* \,|\, \mathbf{X}, \boldsymbol{\theta}\right)$ in Eq. (B.5) can be written as $\frac{\partial}{\partial\theta_k} \log p_{\mathcal{N}}\left(\mathbf{Y}^* \,|\, \mathbf{X}, \boldsymbol{\theta}\right) = -\frac{\partial}{\partial\theta_k} f(\theta_k) \frac{\partial}{\partial\theta_k} E_{\mathbf{X},\mathbf{Y}^*}(\boldsymbol{\theta})$ which for our choice of $f(\theta_k)$, Eqs. (B.6), becomes

$$\frac{\partial}{\partial\theta_k} \log p_{\mathcal{N}}\left(\mathbf{Y}^* \,|\, \mathbf{X}, \boldsymbol{\theta}\right) = -\sigma(\gamma\, s_k\, \theta_k)\, s_k \frac{\partial}{\partial\theta_k} E_{\mathbf{X},\mathbf{Y}^*}(\boldsymbol{\theta}) \,, \tag{B.8}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ denotes the sigmoid function. The error gradient $\frac{\partial}{\partial\theta_k} E_{\mathbf{X},\mathbf{Y}^*}(\boldsymbol{\theta})$ can be computed using standard Error Backpropagation Neal, 1992; Rumelhart et al., 1985.

Theorem 1 requires that Eq. (B.8) is twice differentiable, which is true for any finite value for γ. In our simulations we used the limiting case of large γ such that dormant connections are actually mapped to zero weight. In this limit, one approaches the simple expression

$$\frac{\partial}{\partial\theta_k} \log p_{\mathcal{N}}\left(\mathbf{Y}^* \,|\, \mathbf{X}, \boldsymbol{\theta}\right) = \begin{cases} 0 & \text{if } \theta_k \leq 0 \\ -s_k \frac{\partial}{\partial\theta_k} E_{\mathbf{X},\mathbf{Y}^*}(\boldsymbol{\theta}) & \text{else} \end{cases} . \tag{B.9}$$

Thus, the gradient (B.9) vanishes for dormant connections ($\theta_k < 0$). Therefore changes of dormant connections are independent of the error gradient.

This leads to the parameter updates of the soft-DEEP R algorithm given by Eq. (B.4). The term $\sqrt{2T\eta}\ \nu_k$ results from the diffusion term $\mathcal{W}_k$ integrated over Δt, where ν_k is a Gaussian random variable with zero mean and unit variance. The term $-\eta\alpha$ results from the exponential prior distribution $p_{\mathcal{S}}\left(\boldsymbol{\theta}\right)$ (the ℓ_1-regularization). Note that this prior is not differentiable at 0. In (B.4) we approximate the gradient by assuming it to be zero at $\theta_k = 0$ and below. Thus, parameters on the negative axis are only driven by a random walk and parameter values might therefore diverge to $-\infty$. To fix this problem we introduced a reflecting boundary at $\theta_{\min}$ (parameters were clipped at this value). Another potential solution would be to use a different prior distribution that also effects the negative axis, however we found that Eq. (B.4) produces very good results in practice.

B.4 Analysis of convergence of the DEEP R algorithm

Here we provide additional details to the convergence properties of the DEEP R algorithm. To do so we formulate the algorithm in terms of a Markov chain that evolves the parameters $\boldsymbol{\theta}$ and the connectivity constraints (listed in Algorithm 3). Each application of the Markov transition operators corresponds to one iteration of the DEEP R algorithm. We show that the distribution of parameters and network connectivities over the iterations of DEEP R converges to the stationary distribution Eq. (2.4) that jointly realizes parameter vectors $\boldsymbol{\theta}$ and admissible connectivity constraints.

```
1 given: initial values $\boldsymbol{\theta}'$, $\mathbf{c}'$ with $|\mathbf{c}'| = M$ ;
2 for $i$ in $[1, N_{iterations}]$ do
3     $\boldsymbol{\theta} \sim \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}', \mathbf{c}')$ ;
4     $\mathbf{c} \sim \mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta})$ ;
5     $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$, $\mathbf{c}' \leftarrow \mathbf{c}$ ;
6 end
```

Algorithm 3: A reformulation of Algorithm 1 that is used for the proof in Theorem 2. Markov transition operators $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}', \mathbf{c}')$ and $\mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta})$ are applied for parameter updates in each iteration. The transition operator $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}', \mathbf{c}')$ updates $\boldsymbol{\theta}$ and corresponds to line 3, $\mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta})$ updates the connectivity constraint vector $\mathbf{c}$ and corresponds to lines 4,7 and 8 of Algorithm 1. $\boldsymbol{\theta}'$ and $\mathbf{c}'$ denote the parameter vector and connectivity constraint of the previous time step, respectively.

Each iteration of DEEP R corresponds to two update steps, which we formally describe in Algorithm 3 using the Markov transition operators $\mathcal{T}_{\boldsymbol{\theta}}$ and $\mathcal{T}_{\mathbf{c}}$ and the binary constraint vector $\mathbf{c} \in \{0,1\}^M$ over all M connections of the network with elements c_k, where $c_k = 1$ represents an active connection k. $\mathbf{c}$ is a constraint on the dynamics, i.e., all connections k for which $c_k = 0$ have to be dormant in the evolution of the parameters. The transition operators are conditional probability distributions from which in each iteration new samples for $\boldsymbol{\theta}$ and $\mathbf{c}$ are drawn for given previous values $\boldsymbol{\theta}'$ and $\mathbf{c}'$.

1. *Parameter update*: The transition operator $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}', \mathbf{c}')$ updates all parameters θ_k for which $c_k = 1$ (active connections) and leaves the parameters θ_k at their current value for $c_k = 0$ (dormant connections). The update of active connections is realized by advancing the SDE (2.2) for an arbitrary time step Δt (line 3 of Algorithm 3).
2. *Connectivity update*: for all parameters θ_k that are dormant, set $c_k = 0$ and randomly select an element c_l which is currently 0 and set it to 1. This corresponds to line 3 of Algorithm 3 and is realized by drawing a new $\mathbf{c}$ from $\mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta})$.

The constraint imposed by $\mathbf{c}$ on $\boldsymbol{\theta}$ is formalized through the deterministic binary function $\mathcal{C}(\boldsymbol{\theta}, \mathbf{c}) \in \{0,1\}$ which is 1 if the parameters $\boldsymbol{\theta}$ are compatible with the constraint vector $\mathbf{c}$ and 0 otherwise. This is expressed as (with $\Rightarrow$ denoting the Boolean implication):

$$\mathcal{C}(\boldsymbol{\theta}, \mathbf{c}) = \begin{cases} 1 & \text{if for all } k, 1 \leq k \leq K: \; c_k = 0 \;\Rightarrow\; \theta_k < 0 \\ 0 & \text{else} \end{cases} . \tag{B.10}$$

The constraint $\mathcal{C}(\boldsymbol{\theta}, \mathbf{c})$ is fulfilled if all connections k with $c_k = 0$ are dormant ($\theta_k < 0$).

Note that the transition operator $\mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta})$ depends only on the parameter vector $\boldsymbol{\theta}$. It samples a new $\mathbf{c}$ with uniform probability among the constraint vectors that are compatible with the current set of parameters $\boldsymbol{\theta}$. We write the number of possible vectors $\mathbf{c}$ that are compatible with $\boldsymbol{\theta}$ as $\mu(\boldsymbol{\theta})$, given by the binomial coefficient (the number of possible selections that fulfill the constraint of new active connections)

$$\mu(\boldsymbol{\theta}) \;=\; \sum_{\mathbf{c}\in\chi} \mathcal{C}(\boldsymbol{\theta},\mathbf{c}) \;=\; \binom{M-|\boldsymbol{\theta}\geq 0|}{K-|\boldsymbol{\theta}\geq 0|}, \quad \text{with} \quad \chi = \left\{ \boldsymbol{\xi} \in \{0,1\}^M \;\middle|\; |\boldsymbol{\xi}| = K \right\}, \tag{B.11}$$

where $|\mathbf{c}|$ denotes the number of non-zero elements in $\mathbf{c}$ and χ is the set of all binary vectors with exactly K elements of value 1. Using this we can define the operator $\mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta})$ as:

$$\mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta}) \;=\; \frac{1}{\mu(\boldsymbol{\theta})} \sum_{\boldsymbol{\xi}\in\chi} \delta(\mathbf{c}-\boldsymbol{\xi})\; \mathcal{C}(\boldsymbol{\theta},\mathbf{c}) \tag{B.12}$$

where δ denotes the vectorized Kronecker delta function, with $\delta(\mathbf{0}) = 1$ and 0 else. Note that Eq. (B.12) assigns non-zero probability only to vectors $\mathbf{c}$ that are zero for elements k for which $\theta_k < 0$ is true (assured by the second term). In addition vectors $\mathbf{c}$ have to fulfill $|\mathbf{c}| = K$. Therefore, sampling from this operator introduces randomly new connection for the number of missing ones in $\boldsymbol{\theta}$. This process models the *connectivity update* of Algorithm 3.

The transition operator $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}',\mathbf{c}')$ in Eq. (B.30) evolves the parameter vector $\boldsymbol{\theta}$ under the constraint $\mathbf{c}$, i.e., it produces parameters confined to the connectivity constraint. By construction this operator has a stationary distribution that is given by the following Lemma.

Lemma 1. *Let $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}',\mathbf{c})$ be the transition operator of the Markov chain over $\boldsymbol{\theta}$ which is defined, as the integration of the SDE written in Eq. (2.2) over an interval Δt for active connections ($c_k = 1$), and as the identity for the remaining dormant connections ($c_k = 0$). Then it leaves the following distribution $p^*(\boldsymbol{\theta}|\mathbf{c})$ invariant*

$$p^*(\boldsymbol{\theta}|\mathbf{c}) = \frac{1}{p^*(\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0})} p^*(\boldsymbol{\theta})\mathcal{C}(\boldsymbol{\theta},\mathbf{c})\,, \tag{B.13}$$

where $\boldsymbol{\theta}_{\in\mathbf{c}}$ denotes the truncation of the vector $\boldsymbol{\theta}$ to the active connections ($c_k = 1$), thus $p^*(\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0})$ is the probability that all connections outside of $\mathbf{c}$ are dormant according to the posterior, and $p^*(\boldsymbol{\theta})$ is the posterior (see Theorem 1).

The proof is divided into two sub proofs. First we show that the distribution defined as $p^*(\boldsymbol{\theta}|\mathbf{c}) = \frac{1}{\mathcal{L}(\mathbf{c})} p^*(\boldsymbol{\theta})\mathcal{C}(\boldsymbol{\theta},\mathbf{c})$ with $\mathcal{L}(\mathbf{c})$ a normalization constant, is left invariant by $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}',\mathbf{c})$, second we will show that this normalization constant has to be equal to $p^*(\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0})$. In coherence with the notation $\boldsymbol{\theta}_{\in\mathbf{c}}$ we will use verbally that θ_k is an element of $\mathbf{c}$ if $c_k = 1$.

Proof. To show that the distribution defined as $p^*(\boldsymbol{\theta}|\mathbf{c}) = \frac{1}{\mathcal{L}(\mathbf{c})} p^*(\theta)\mathcal{C}(\boldsymbol{\theta},\mathbf{c})$ is left invariant, we will show directly that $\int_{\boldsymbol{\theta}'} \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}',\mathbf{c}) p^*(\boldsymbol{\theta}'|\mathbf{c}) d\boldsymbol{\theta}' = p^*(\boldsymbol{\theta}|\mathbf{c})$. To do so we will show that both $p^*(\boldsymbol{\theta}'|\mathbf{c})$ and $\mathcal{T}$ factorizes in terms that depend only on $\boldsymbol{\theta}'_{\in\mathbf{c}}$

or on $\boldsymbol{\theta}'_{\notin \mathbf{c}}$ and thus we will be able to separate the integral over $\boldsymbol{\theta}'$ as the product of two simpler integrals.

We first study the distribution $p^*(\boldsymbol{\theta}'_{\in \mathbf{c}}|\mathbf{c})$. Before factorizing, one has to notice a strong property of this distribution. Let's partition the tempered posterior distribution $p^*(\boldsymbol{\theta}')$ over the cases when the constraint is satisfied or not

$$\begin{aligned} p^*(\boldsymbol{\theta}'|\mathbf{c}) &= \frac{1}{\mathcal{L}(\mathbf{c})} p^*(\boldsymbol{\theta}')\mathcal{C}(\mathbf{c},\boldsymbol{\theta}) & \text{(B.14)} \\ &= \frac{1}{\mathcal{L}(\mathbf{c})} \left[p^*(\boldsymbol{\theta}',\mathcal{C}(\mathbf{c},\boldsymbol{\theta})=1) + p^*(\boldsymbol{\theta}',\mathcal{C}(\mathbf{c},\boldsymbol{\theta})=0) \right] \mathcal{C}(\mathbf{c},\boldsymbol{\theta}) & \text{(B.15)} \end{aligned}$$

when we multiply individually the first and the second term with $\mathcal{C}(\mathbf{c},\boldsymbol{\theta})$, $\mathcal{C}(\mathbf{c},\boldsymbol{\theta})$ can be replaced by its binary value and the second term is always null. It remains that

$$p^*(\boldsymbol{\theta}'|\mathbf{c}) = \frac{1}{\mathcal{L}(\mathbf{c})} p^*(\boldsymbol{\theta}',\mathcal{C}(\mathbf{c},\boldsymbol{\theta})=1) \qquad \text{(B.16)}$$

seeing that one can rewrite the condition $\mathcal{C}(\mathbf{c},\boldsymbol{\theta}) = 1$ as the condition on the sign of the random variable $\boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0}$ (note that in this inequality $\mathbf{c}$ is a deterministic constant and $\boldsymbol{\theta}'_{\notin \mathbf{c}}$ is a random variable)

$$p^*(\boldsymbol{\theta}'|\mathbf{c}) = \frac{1}{\mathcal{L}(\mathbf{c})} p^*(\boldsymbol{\theta}',\boldsymbol{\theta}'_{\notin \mathbf{c}} < \mathbf{0}) \qquad \text{(B.17)}$$

We can factorize the conditioned posterior as $p^*(\boldsymbol{\theta},\boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0}) = p^*(\boldsymbol{\theta}_{\in \mathbf{c}}|\boldsymbol{\theta}_{\notin \mathbf{c}},\boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0})p^*(\boldsymbol{\theta}_{\notin \mathbf{c}},\boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0})$. But when the dormant parameters are negative $\boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0}$, the active parameters $\boldsymbol{\theta}_{\in \mathbf{c}}$ do not depend on the actual value of the dormant parameters $\boldsymbol{\theta}_{\notin \mathbf{c}}$, so we can simplify the conditions of the first factor further to obtain

$$p^*(\boldsymbol{\theta}'|\mathbf{c}) = \frac{1}{\mathcal{L}(\mathbf{c})} p^*(\boldsymbol{\theta}'_{\in \mathbf{c}}|\boldsymbol{\theta}'_{\notin \mathbf{c}} < \mathbf{0}) p^*(\boldsymbol{\theta}'_{\notin \mathbf{c}},\boldsymbol{\theta}'_{\notin \mathbf{c}} < \mathbf{0}) \qquad \text{(B.18)}$$

We now study the operator $\mathcal{T}_{\boldsymbol{\theta}}$. It factorizes similarly because it is built out of two independent operations: one that integrates the SDE over the active connections and one that applies identity to the dormant ones. Moreover all the terms in the SDE which evolve the active parameters $\boldsymbol{\theta}_{\notin \mathbf{c}}$ are independent of the dormant ones $\boldsymbol{\theta}_{\notin \mathbf{c}}$ as long as we know they are dormant. Thus, the operator $\mathcal{T}_{\boldsymbol{\theta}}$ splits in two

$$\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}',\mathbf{c}) = \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}_{\in \mathbf{c}}|\boldsymbol{\theta}'_{\in \mathbf{c}},\mathbf{c})\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}_{\notin \mathbf{c}}|\boldsymbol{\theta}'_{\notin \mathbf{c}},\mathbf{c}) \qquad \text{(B.19)}$$

To finally separate the integration over $\boldsymbol{\theta}$ as a product of two integrals we need to make sure that all the factor depend only on the variable $\boldsymbol{\theta}'_{\in \mathbf{c}}$ or only on $\boldsymbol{\theta}'_{\notin \mathbf{c}}$. This might not seem obvious but even the conditioned probability $p^*(\boldsymbol{\theta}'_{\in \mathbf{c}}|\boldsymbol{\theta}'_{\notin \mathbf{c}} < \mathbf{0})$ is a function of $\boldsymbol{\theta}'_{\in \mathbf{c}}$ because in the conditioning $\boldsymbol{\theta}'_{\notin \mathbf{c}} < \mathbf{0}$, $\boldsymbol{\theta}'_{\notin \mathbf{c}}$ refers to the random variable and not to a specific value over which we integrate. As a result the double integral is equal to the product of the two integrals

$$\begin{aligned} \int_{\boldsymbol{\theta}'} \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}',\mathbf{c}) p^*(\boldsymbol{\theta}'|\mathbf{c}) d\boldsymbol{\theta}' &= \frac{1}{\mathcal{L}(\mathbf{c})} \int_{\boldsymbol{\theta}'_{\in \mathbf{c}}} \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}_{\in \mathbf{c}}|\boldsymbol{\theta}'_{\in \mathbf{c}},\mathbf{c}) p^*(\boldsymbol{\theta}'_{\in \mathbf{c}}|\boldsymbol{\theta}'_{\notin \mathbf{c}} < \mathbf{0}) d\boldsymbol{\theta}'_{\in \mathbf{c}} & \text{(B.20)} \\ & \int_{\boldsymbol{\theta}'_{\notin \mathbf{c}}} \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}_{\notin \mathbf{c}}|\boldsymbol{\theta}'_{\notin \mathbf{c}},\mathbf{c}) p^*(\boldsymbol{\theta}'_{\notin \mathbf{c}},\boldsymbol{\theta}'_{\notin \mathbf{c}} < \mathbf{0}) d\boldsymbol{\theta}'_{\notin \mathbf{c}} & \text{(B.21)} \end{aligned}$$

We can now study the two integrals separately. The second integral over the parameters $\boldsymbol{\theta}_{\notin \mathbf{c}}$ is simpler because by construction the operator $\mathcal{T}_{\boldsymbol{\theta}}$ is the identity

$$\int_{\boldsymbol{\theta}'_{\notin \mathbf{c}}} \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}_{\notin \mathbf{c}} | \boldsymbol{\theta}'_{\notin \mathbf{c}}, \mathbf{c}) p^*(\boldsymbol{\theta}'_{\notin \mathbf{c}}, \boldsymbol{\theta}'_{\notin \mathbf{c}} < \mathbf{0}) d\boldsymbol{\theta}'_{\notin \mathbf{c}} \quad = \quad p^*(\boldsymbol{\theta}_{\notin \mathbf{c}}, \boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0}) \tag{B.22}$$

There is more to say about the first integral over the active connections $\boldsymbol{\theta}_{\in \mathbf{c}}$. The operator $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}_{\in \mathbf{c}} | \boldsymbol{\theta}'_{\notin \mathbf{c}}, \mathbf{c})$ integrates over the active parameters $\boldsymbol{\theta}_{\in \mathbf{c}}$ the same SDE as before with the difference that the network is reduced to a sparse architecture where only the parameters $\boldsymbol{\theta}_{\in \mathbf{c}}$ are active. We want to find the relationship between the stationary distribution of this new operator and $p^*(\boldsymbol{\theta})$ that is written in the integral which is defined in equation (2.3) as the tempered posterior of the dense network. In fact, the tempered posterior of the dense network marginalized and conditioned over the dormant connections $p^*(\boldsymbol{\theta}'_{\in \mathbf{c}} | \boldsymbol{\theta}'_{\notin \mathbf{c}} < \mathbf{0})$ is equal to the stationary distribution of $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}_{\in \mathbf{c}} | \boldsymbol{\theta}'_{\notin \mathbf{c}}, \mathbf{c})$ (i.e. of the SDE in the sparse network). To prove this, we detail in the following paragraph that the drift in the SDE evolving the sparse network is given by the log-posterior of the dense network condition on $\boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0}$ and using Theorem 1, we will conclude that $p^*(\boldsymbol{\theta}'_{\in \mathbf{c}} | \boldsymbol{\theta}'_{\notin \mathbf{c}} < \mathbf{0})$ is the stationary distribution of $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}_{\in \mathbf{c}} | \boldsymbol{\theta}'_{\notin \mathbf{c}}, \mathbf{c})$.

We write the prior and the likelihood of the sparse network as function of the prior and the likelihood $p_{\mathcal{S}}$ with $p_{\mathcal{N}}$ of the dense network. The likelihood in the sparse network is defined as previously with the exception that the dormant connections are given zero-weight $w_k = 0$ so it is equal to $p_{\mathcal{N}}(\mathbf{X}, \mathbf{Y}^* | \boldsymbol{\theta}_{\in \mathbf{c}}, \boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0})$. The difference between the prior that defines soft-DEEP R and the prior of DEEP R remains in the presence of the constraint. When considering the sparse network defined by $\mathbf{c}$ the constraint is satisfied and the prior of soft-DEEP R marginalized over the dormant connections $p_{\mathcal{S}}(\boldsymbol{\theta}_{\in \mathbf{c}})$ is the prior of the sparse network with $p_{\mathcal{S}}$ defined as before. As this prior is connection-specific ($p_{\mathcal{S}}(\theta_i)$ independent of θ_j), this implies that $p_{\mathcal{S}}(\boldsymbol{\theta}_{\in \mathbf{c}})$ is independent of the dormant connection, and the prior $p_{\mathcal{S}}(\boldsymbol{\theta}_{\in \mathbf{c}})$ is equal to $p_{\mathcal{S}}(\boldsymbol{\theta}_{\in \mathbf{c}} | \boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0})$. Thus, we can write the posterior of the sparse network which is by definition proportional to the product $p_{\mathcal{N}}(\mathbf{X}, \mathbf{Y}^* | \boldsymbol{\theta}_{\in \mathbf{c}}, \boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0}) p_{\mathcal{S}}(\boldsymbol{\theta}_{\in \mathbf{c}} | \boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0})$. Looking back to the definition of the posterior of the dense network this product is actually proportional to posterior of the dense network conditioned on the negativity of dormant connections $p^*(\boldsymbol{\theta}_{\in \mathbf{c}} | \boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0}, \mathbf{X}, \mathbf{Y}^*)$. The posterior of the sparse network is therefore proportional to the conditioned posterior of the dense network but as they both normalize to 1 they are actually equal. Writing down the new SDE, the diffusion term $\sqrt{2T\beta d\mathcal{W}_k}$ remains unchanged, and the drift term is given by the gradient of the log-posterior $\log p^*(\boldsymbol{\theta}_{\in \mathbf{c}} | \boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0}, \mathbf{X}, \mathbf{Y}^*)$. Applying Theorem 1 to this new SDE, we now confirm that the tempered and conditioned posterior of the dense network $p^*(\boldsymbol{\theta}_{\in \mathbf{c}} | \boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0})$ is left invariant by the SDE evolving the sparse network. As $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}_{\in \mathbf{c}} | \boldsymbol{\theta}'_{\notin \mathbf{c}}, \mathbf{c})$ is the integration for a given Δt of this SDE, it also leaves $p^*(\boldsymbol{\theta}_{\in \mathbf{c}} | \boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0})$ invariant. This yields

$$\int_{\boldsymbol{\theta}'_{\in \mathbf{c}}} \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}_{\in \mathbf{c}} | \boldsymbol{\theta}'_{\in \mathbf{c}}, \mathbf{c}) p^*(\boldsymbol{\theta}'_{\in \mathbf{c}} | \boldsymbol{\theta}'_{\notin \mathbf{c}} < \mathbf{0}) d\boldsymbol{\theta}'_{\in \mathbf{c}} \quad = \quad p^*(\boldsymbol{\theta}_{\in \mathbf{c}} | \boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0}) \tag{B.23}$$

As we simplified both integrals we arrived at

$$\int_{\boldsymbol{\theta}'} \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}',\mathbf{c})p^*(\boldsymbol{\theta}'|\mathbf{c})d\boldsymbol{\theta}' = \frac{1}{\mathcal{L}(\mathbf{c})}p^*(\boldsymbol{\theta}_{\in\mathbf{c}}|\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0})p^*(\boldsymbol{\theta}_{\notin\mathbf{c}},\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0}) \quad \text{(B.24)}$$

Replacing the right-end side with equation (B.18) we conclude

$$\int_{\boldsymbol{\theta}'} \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}',\mathbf{c})p^*(\boldsymbol{\theta}'|\mathbf{c})d\boldsymbol{\theta}' = p^*(\boldsymbol{\theta}|\mathbf{c}) \quad \text{(B.25)}$$

□

We now show that the normalization constant $\mathcal{L}(\mathbf{c})$ is equal to $p^*(\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0})$.

Proof. Using equation (B.18), as $p^*(\theta|\mathbf{c})$ normalizes to 1 the normalization constant is equal to

$$\mathcal{L}(\mathbf{c}) = \int_{\boldsymbol{\theta}} p^*(\boldsymbol{\theta}_{\in\mathbf{c}}|\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0})p^*(\boldsymbol{\theta}_{\notin\mathbf{c}},\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0})d\boldsymbol{\theta} \quad \text{(B.26)}$$

By factorizing the last factor in the integral we have that

$$p^*(\boldsymbol{\theta},\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0}) = p^*(\boldsymbol{\theta}_{\in\mathbf{c}}|\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0})p^*(\boldsymbol{\theta}_{\notin\mathbf{c}}|\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0})p^*(\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0}) \quad \text{(B.27)}$$

The last term does not depend on the value $\boldsymbol{\theta}$ because $\boldsymbol{\theta}_{\notin\mathbf{c}}$ refers here to the random variable and the first two term depend either on $\boldsymbol{\theta}_{\in\mathbf{c}}$ or $\boldsymbol{\theta}_{\notin\mathbf{c}}$. Plugging the previous equation into the computation of $\mathcal{L}(\mathbf{c})$ and separating the integrals we have

$$\mathcal{L}(\mathbf{c}) = p^*(\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0}) \underbrace{\int_{\boldsymbol{\theta}_{\in\mathbf{c}}} p^*(\boldsymbol{\theta}_{\in\mathbf{c}}|\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0})d\boldsymbol{\theta}_{\in\mathbf{c}}}_{=1} \underbrace{\int_{\boldsymbol{\theta}_{\notin\mathbf{c}}} p^*(\boldsymbol{\theta}_{\notin\mathbf{c}}|\boldsymbol{\theta}_{\notin\mathbf{c}} < \mathbf{0})d\boldsymbol{\theta}_{\notin\mathbf{c}}}_{=1} \quad \text{(B.28)}$$

□

Due to Lemma 1, there exists a distribution $\pi(\boldsymbol{\theta}\,|\,\mathbf{c})$ of the following form which is left invariant by the operator $\mathcal{T}_{\boldsymbol{\theta}}$

$$\pi(\boldsymbol{\theta}\,|\,\mathbf{c}) = \frac{1}{\mathcal{L}(\mathbf{c})}\pi(\boldsymbol{\theta})\,\mathcal{C}(\boldsymbol{\theta},\mathbf{c})\,, \quad \text{(B.29)}$$

where $\mathcal{L}(\mathbf{c})$ is a normalizer and where $\pi(\boldsymbol{\theta})$ is some distribution over $\boldsymbol{\theta}$ that may not obey the constraint $\mathcal{C}(\boldsymbol{\theta},\mathbf{c})$. This will imply a very strong property on the compound operator which evolves both $\boldsymbol{\theta}$ and $\mathbf{c}$. To form $\mathcal{T}$ the operators $\mathcal{T}_{\boldsymbol{\theta}}$ and $\mathcal{T}_{\mathbf{c}}$ are performed one after the other so that the total update can be written in terms of the compound operator

$$\mathcal{T}(\boldsymbol{\theta},\mathbf{c}|\boldsymbol{\theta}',\mathbf{c}') = \mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta})\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}',\mathbf{c}')\,. \quad \text{(B.30)}$$

Applying the compound operator $\mathcal{T}$ given by Eq. (B.30) corresponds to advancing the parameters for a single iteration of Algorithm 3.

Using these definitions a general theorem can be enunciated for arbitrary distributions $\pi(\boldsymbol{\theta} \mid \mathbf{c})$ of the form (B.29). The following theorem states that the distribution of variable pairs $\mathbf{c}$ and $\boldsymbol{\theta}$ that is left stationary by the operator $\mathcal{T}$ is the product of Eq. (B.29) and a uniform prior $p_{\mathcal{C}}(\mathbf{c})$ over the constraint vectors which have K active connections. This prior is formally defined as

$$p_{\mathcal{C}}(\mathbf{c}) \;=\; \frac{1}{|\chi|} \sum_{\boldsymbol{\xi} \in \chi} \delta(\mathbf{c} - \boldsymbol{\xi}) \,, \tag{B.31}$$

with χ as defined in (B.11). The theorem to analyze the dynamics of Algorithm 3 can now be written as

Theorem 2. *Let $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}', \mathbf{c})$ be the transition operator of a Markov chain over $\boldsymbol{\theta}$ and let $\mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta})$ be defined by Eq.* (B.12)*. Under the assumption that $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}', \mathbf{c})$ has a unique stationary distribution $\pi(\boldsymbol{\theta}|\mathbf{c})$, that verifies Eq.* (B.29)*, then the Markov chain over $\boldsymbol{\theta}$ and $\mathbf{c}$ with transition operator*

$$\mathcal{T}(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\theta}', \mathbf{c}') = \mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta})\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}', \mathbf{c}') \tag{B.32}$$

leaves the stationary distribution

$$p^*(\boldsymbol{\theta}, \mathbf{c}) = \frac{\mathcal{L}(\mathbf{c})}{\sum_{\mathbf{c}' \in \mathcal{X}} \mathcal{L}(\mathbf{c}')} \pi(\boldsymbol{\theta}|\mathbf{c}) p_{\mathcal{C}}(\mathbf{c}) \tag{B.33}$$

invariant. If the Markov chain of the transition operator $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}', \mathbf{c}')$ is ergodic, then the stationary distribution is also unique.

Proof. Theorem 2 holds for $\mathcal{T}_{\mathbf{c}}$ in combination with any operator $\mathcal{T}_{\boldsymbol{\theta}}$ that updates $\boldsymbol{\theta}$ that can be written in the form (B.29). We prove Theorem 2 by proving the following equality to show that $\mathcal{T}$ leaves (B.33) invariant:

$$\sum_{\mathbf{c}'} \int \mathcal{T}(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\theta}', \mathbf{c}')\, p^*(\boldsymbol{\theta}', \mathbf{c}')\, \mathrm{d}\boldsymbol{\theta}' \;=\; p^*(\boldsymbol{\theta}, \mathbf{c}) \,. \tag{B.34}$$

We expand the left-hand term using Eq. (B.32) and Eq. (B.33)

$$\begin{aligned} &\sum_{\mathbf{c}'} \int \mathcal{T}(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\theta}', \mathbf{c}')\, p^*(\boldsymbol{\theta}', \mathbf{c}')\, \mathrm{d}\boldsymbol{\theta}' \;= \\ \sum_{\mathbf{c}'} \int \mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta})&\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}', \mathbf{c}')\, \frac{\mathcal{L}(\mathbf{c}')}{\sum_{\mathbf{c}'' \in \mathcal{X}} \mathcal{L}(\mathbf{c}'')}\, \pi(\boldsymbol{\theta}'|\mathbf{c}') p_{\mathcal{C}}(\mathbf{c}')\, \mathrm{d}\boldsymbol{\theta}' \,. \end{aligned} \tag{B.35}$$

Since $\mathcal{T}_{\mathbf{c}}$ does not depend on $\boldsymbol{\theta}'$ and $\mathbf{c}'$, one can pull it out of the sum and integral and then marginalize over $\boldsymbol{\theta}'$ by observing that $\pi(\boldsymbol{\theta}|\mathbf{c}')$ is by definition the stationary distribution of $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}', \mathbf{c}')$:

$$\sum_{\mathbf{c}'} \int \mathcal{T}(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\theta}', \mathbf{c}')\, p^*(\boldsymbol{\theta}', \mathbf{c}')\, \mathrm{d}\boldsymbol{\theta}' \;= \tag{B.36}$$

$$= \mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta}) \sum_{\mathbf{c}'} \int \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}', \mathbf{c}')\, \frac{\mathcal{L}(\mathbf{c}')}{\sum_{\mathbf{c}''\in\mathcal{X}} \mathcal{L}(\mathbf{c}'')}\, \pi(\boldsymbol{\theta}'|\mathbf{c}') p_{\mathcal{C}}(\mathbf{c}')\, \mathrm{d}\boldsymbol{\theta}' \tag{B.37}$$

$$= \mathcal{T}_{\mathbf{c}}(\mathbf{c}|\boldsymbol{\theta}) \sum_{\mathbf{c}'} \frac{\mathcal{L}(\mathbf{c}')}{\sum_{\mathbf{c}''\in\mathcal{X}} \mathcal{L}(\mathbf{c}'')} \pi(\boldsymbol{\theta}|\mathbf{c}') p_{\mathcal{C}}(\mathbf{c}')\,. \tag{B.38}$$

What remains to be done is to marginalize over $\mathbf{c}'$ and to relate the result to the stationary distribution $p^*(\boldsymbol{\theta}, \mathbf{c}) = \frac{\mathcal{L}(\mathbf{c})}{\sum_{\mathbf{c}'\in\mathcal{X}} \mathcal{L}(\mathbf{c}')} \pi(\boldsymbol{\theta}|\mathbf{c}) p_{\mathcal{C}}(\mathbf{c})$. First we replace $\mathcal{T}_{\mathbf{c}}$ with its definition Eq. (B.12):

$$\sum_{\mathbf{c}'} \int \mathcal{T}(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\theta}', \mathbf{c}')\, p^*(\boldsymbol{\theta}', \mathbf{c}')\, \mathrm{d}\boldsymbol{\theta}' \;=$$

$$= \left(\frac{1}{\mu(\boldsymbol{\theta})} \sum_{\boldsymbol{\xi}\in\boldsymbol{\chi}} \delta(\mathbf{c} - \boldsymbol{\xi}) \right) \mathcal{C}(\boldsymbol{\theta}, \mathbf{c}) \sum_{\mathbf{c}'} \frac{\mathcal{L}(\mathbf{c}')}{\sum_{\mathbf{c}''\in\mathcal{X}} \mathcal{L}(\mathbf{c}'')} \pi(\boldsymbol{\theta}|\mathbf{c}') p_{\mathcal{C}}(\mathbf{c}')$$

As the operator $\mathcal{T}_{\mathbf{c}}$ samples uniform across admissible configurations it has a close relationship with the uniform probability distribution $p_{\mathcal{C}}$ and we can now replace the sum over $\boldsymbol{\xi}$ using Eq. (B.31)

$$\sum_{\mathbf{c}'} \int \mathcal{T}(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\theta}', \mathbf{c}')\, p^*(\boldsymbol{\theta}', \mathbf{c}')\, \mathrm{d}\boldsymbol{\theta}' \;=\; \frac{|\boldsymbol{\chi}|}{\mu(\boldsymbol{\theta})} p_{\mathcal{C}}(\mathbf{c})\, \mathcal{C}(\boldsymbol{\theta}, \mathbf{c}) \sum_{\mathbf{c}'} \frac{\mathcal{L}(\mathbf{c}')}{\sum_{\mathbf{c}''\in\mathcal{X}} \mathcal{L}(\mathbf{c}'')} \pi(\boldsymbol{\theta}|\mathbf{c}') p_{\mathcal{C}}(\mathbf{c}')\,.$$

From Eq. (B.11), Eq. (B.29) and Eq. (B.31) we find the equalities $\sum_{\mathbf{c}'} \mathcal{L}(\mathbf{c}')\pi(\boldsymbol{\theta}|\mathbf{c}') p_{\mathcal{C}}(\mathbf{c}') = \pi(\boldsymbol{\theta}) \sum_{\mathbf{c}'} \mathcal{C}(\boldsymbol{\theta}, \mathbf{c}') p_{\mathcal{C}}(\mathbf{c}') = \frac{\mu(\boldsymbol{\theta})}{|\boldsymbol{\chi}|} \pi(\boldsymbol{\theta})$. Using this we get

$$\sum_{\mathbf{c}'} \int \mathcal{T}(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\theta}', \mathbf{c}')\, p^*(\boldsymbol{\theta}', \mathbf{c}')\, \mathrm{d}\boldsymbol{\theta}' \;=\; \frac{|\boldsymbol{\chi}|}{\mu(\boldsymbol{\theta})} p_{\mathcal{C}}(\mathbf{c})\, \mathcal{C}(\boldsymbol{\theta}, \mathbf{c})\, \frac{1}{\sum_{\mathbf{c}'\in\mathcal{X}} \mathcal{L}(\mathbf{c}')} \frac{\mu(\boldsymbol{\theta})}{|\boldsymbol{\chi}|} \pi(\boldsymbol{\theta})$$

Finally using again Eq. (B.29), i.e. $\pi(\boldsymbol{\theta})\, \mathcal{C}(\boldsymbol{\theta}, \mathbf{c}) = \mathcal{L}(\mathbf{c}) \pi(\boldsymbol{\theta}|\mathbf{c})$

$$\sum_{\mathbf{c}'} \int \mathcal{T}(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\theta}', \mathbf{c}')\, p^*(\boldsymbol{\theta}', \mathbf{c}')\, \mathrm{d}\boldsymbol{\theta}' \;=\; \frac{\mathcal{L}(\mathbf{c})}{\sum_{\mathbf{c}'\in\mathcal{X}} \mathcal{L}(\mathbf{c}')} \pi(\boldsymbol{\theta}|\mathbf{c})\, p_{\mathcal{C}}(\mathbf{c}) \;=\; p^*(\boldsymbol{\theta}, \mathbf{c})\,.$$

This shows that the stationary distribution Eq. (B.33) is invariant under the compound operator (B.32). Under the assumption that $\mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\theta}', \mathbf{c}')$ is ergodic it allows each parameter θ_k to become negative with non-zero probability and the stationary distribution is also unique. This can be seen by noting that under this assumption each connection will become dormant sooner or later and thus each state in $\mathbf{c}$ can be reached from any other state $\mathbf{c}'$. The Markov chain is therefore irreducible and the stationary distribution is unique. $\square$

Lemma 1 provides for the case of algorithm 3 the existence of an invariant distribution that is needed to apply Theorem 2. We conclude that the distribution $p^*(\boldsymbol{\theta}, \mathbf{c})$ defined by plugging the result of Lemma 1 Eq. (B.13) into the result of Theorem 2 Eq. (B.33), is left invariant by algorithm 3 and it is written

$$\boxed{p^*(\boldsymbol{\theta}, \mathbf{c}) = \frac{p^*(\boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0})}{\sum_{\mathbf{c}' \in \mathcal{X}} p^*(\boldsymbol{\theta}_{\notin \mathbf{c}'} < \mathbf{0})} p^*(\boldsymbol{\theta}|\mathbf{c}) p_{\mathcal{C}}(\mathbf{c})} \tag{B.39}$$

Where $p^*(\boldsymbol{\theta})$ is defined as previously as the tempered posterior of the dense network which is left invariant by soft-DEEP R according to Theorem 1. The prior $p_{\mathcal{C}}(\mathbf{c})$ in Eq. (B.39) assures that only constraints $\mathbf{c}$ with exactly K active connections are selected. By Theorem 2 the stationary distribution (B.39) is also unique. By inserting the result of Lemma 1, Eq. (B.13) we recover Eq. 2.4 of the main text.

Interestingly, by marginalizing over $\boldsymbol{\theta}$, we can show that the network architecture identified by $\mathbf{c}$ is sampled by algorithm 3 from the probability distribution

$$\boxed{p^*(\mathbf{c}) = \frac{p^*(\boldsymbol{\theta}_{\notin \mathbf{c}} < \mathbf{0})}{\sum_{\mathbf{c}' \in \mathcal{X}} p^*(\boldsymbol{\theta}_{\notin \mathbf{c}'} < \mathbf{0})} p_{\mathcal{C}}(\mathbf{c})} \tag{B.40}$$

The difference between the formal algorithm 3 and the actual implementation of DEEP R are that $\mathcal{T}_{\mathbf{c}}$ keeps the dormant connection parameters constant, whereas in DEEP R we implement this by setting connections to 0 as they become negative. We found the process used in DEEP R works very well in practice. The reason why we did not implement algorithm 3 in practice is that we did not want to consume memory by storing any parameter for dormant connections. This difference is obsolete from the view point of the network function for a given $(\boldsymbol{\theta}, \mathbf{c})$ pair because nor negative neither strictly zero θ have any influence on the network function.

This difference might seem problematic to consider that the properties of convergence to a specific stationary distribution as proven for algorithm 3 extends to DEEP R. However, both the theorem and the implementation are rather unspecific regarding the choice of the prior on the negative sides $\theta < 0$. We believe that, with good choices of priors on the negative side, the conceptual and quantitative difference between the distribution explored by 3 and DEEP R are minor, and in general, algorithm 3 is a decent mathematical formalization of DEEP R for the purpose of this paper.

Appendix C

Methods for "Long short-term memory and learning-to-learn in networks of spiking neurons"

C.1 LSNN model

Neuron model: In continuous time the spike trains $x_i(t)$ and $z_j(t)$ are formalized as sums of Dirac pulses. Neurons are modeled according to a standard adaptive leaky integrate-and-fire model. A neuron j spikes as soon at its membrane potential $V_j(t)$ is above its threshold $B_j(t)$. At each spike time t, the membrane potential $V_j(t)$ is reset by subtracting the current threshold value $B_j(t)$ and the neuron enters a strict refractory period where it cannot spike again. Importantly at each spike the threshold $B_j(t)$ of an adaptive neuron is increased by a constant $\beta/\tau_{a,j}$. Then the threshold decays back to a baseline value b_j^0. Between spikes the membrane voltage $V_j(t)$ and the threshold $B_j(t)$ are following the dynamics

$$\begin{aligned}\tau_m \dot{V}_j(t) &= -V_j(t) + R_m I_j(t) &\text{(C.1)}\\ \tau_{a,j}\dot{B}_j(t) &= b_j^0 - B_j(t), &\text{(C.2)}\end{aligned}$$

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

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

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. All network neurons are connected to a population of readout neurons with weights W_{kj}^{out}. When network neuron j spikes, the output synaptic strength W_{kj}^{out} is added to the membrane voltage $y_k(t)$ of all readout neurons k. $y_k(t)$ also follows the dynamics of a leaky integrator $\tau_m \dot{y}_k(t) = -y_k(t)$.

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

$$\begin{aligned} V_j(t+\delta t) &= \alpha V_j(t) + (1-\alpha) R_m I_j(t) - B_j(t) z_j(t) \delta t && \text{(C.4)} \\ b_j(t+\delta t) &= \rho_j b_j(t) + (1-\rho_j) z_j(t), && \text{(C.5)} \end{aligned}$$

where $\alpha = \exp(-\frac{\delta t}{\tau_m})$ and $\rho_j = \exp(-\frac{\delta t}{\tau_{a,j}})$. The term $B_j(t) z_j(t) \delta t$ implements the reset of the membrane voltage after each spike. The current $I_j(t)$ is the weighted sum of the incoming spikes. The definition of the input current in equation (C.3) holds also for discrete time, with the difference that spike trains now assume values in $\{0, \frac{1}{\delta t}\}$.

C.2 Applying BPTT with DEEP R to RSNNs and LSNNs

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

$$\frac{dz_j(t)}{dv_j(t)} := \max\{0, 1 - |v_j(t)|\}, \tag{C.6}$$

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

$$\frac{dz_j(t)}{dv_j(t)} := \gamma \max\{0, 1 - |v_j(t)|\}. \tag{C.7}$$

Note that in adaptive neurons, gradients can propagate through many time steps in the dynamic threshold. This propagation is not affected by the dampening.

Rewiring and weight initialization of excitatory and inhibitory neurons: In all experiments except those reported in Fig. 2, the neurons were either excitatory or inhibitory. When the neuron sign were not constrained, the initial network 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 in the considered weight matrix (i.e., the number of columns of the matrix), $\mathcal{N}(0,1)$ is the zero-mean unit-variance Gaussian distribution and w_0 is a weightscaling factor chosen to be $w_0 = \frac{1\text{Volt}}{R_m}\delta t$. With this choice of w_0 the resistance R_m becomes obsolete but the vanishing-exploding gradient theory (Bengio et al., 1994; Sussillo and L. Abbott, 2014) can be used to avoid tuning by hand the scaling of W_{ji}. In particular the scaling $\frac{1}{\sqrt{n_{in}}}$ used above was sufficient to initialize networks with realistic firing rates and that can be trained efficiently.

When the neuron sign were constrained, all outgoing weights W_{ji}^{rec} or W_{ji}^{out} of a neuron i had the same sign. In those cases, DEEP R Bellec et al., 2018a was used as it maintains the sign of each synapse during training. The sign is thus inherited from the initialization of the network weights. This raises the need of an efficient initialization of weight matrices for given fractions of inhibitory and excitatory neurons. To do so, a sign $\kappa_i \in \{-1,1\}$ is generated randomly for each neuron i by sampling from a Bernoulli distribution. The weight matrix entries are then sampled from $W_{ji} \sim \kappa_i|\mathcal{N}(0,1)|$ and post-processed to avoid exploding gradients. Firstly, a constant is added to each weight so that the sum of excitatory and inhibitory weights onto each neuron j ($\sum_i W_{ji}$) is zero Rajan and L. F. Abbott, 2006 (if j has no inhibitory or no excitatory incoming connections this step is omitted). To avoid exploding gradients it is important to scale the weight so that the largest eigenvalue is lower of equal to 1 (Bengio et al., 1994). Thus, we divided W_{ji} by the absolute value of its largest eigenvalue. When the matrix is not square, eigenvalues are ill-defined. Therefore, we first generated a large enough square matrix and selected the required number of rows or columns with uniform probabilities. The final weight matrix is scaled by w_0 for the same reasons as before.

To initialize matrices with a sparse connectivity, dense matrices were generated as described above and multiplied with a binary mask. The binary mask was generated by sampling uniformly the neuron coordinates that were non-zero at initialization. DEEP R maintains the initial connectivity level throughout training by dynamically disconnecting synapses and reconnecting others elsewhere. The L_1-norm regularization parameter of DEEP R was set to 0.01 and the temperature parameter of DEEP R was left at 0.

C.3 Computational performance of LSNNs

MNIST setup: The pixels of an MNIST image were presented sequentially to the LSNN in 784 time steps. Two input encodings were considered. First, we used a population coding where the grey scale value (which is in the range $[0,1]$) of the currently presented pixel was directly used as the firing probability of each of the 80 input neurons in that time step.

In a second type of input encoding – that is closer to the way how spiking vision sensors encode their input – 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 of the currently presented pixel crossed its threshold. Here, we used two input neurons per threshold, one spiked at threshold crossings from below, and one at the crossings from above. This input convention was chosen for the LSNN results of Fig. 1.B.

The output of the network was determined by averaging the readout output over the 56 time steps following the presentation of the digit. The network was trained by minimizing the cross entropy error between the softmax of the averaged readout and the label distributions. The best performing models use rewiring with a global connectivity level of 12% was used during training to optimize a sparse network connectivity structure (i.e., when randomly picking two neurons in the network, the probability that they would be connected is 0.12). This implies that only a fraction of the parameters were finally used as compared to a similarly performing LSTM network.

Tables C.1 and C.2 contain the results and details of training runs where each time step lasted for 1 ms and 2 ms respectively.

Model	# neurons	conn.	# params	# runs	mean	std.	max.
LSTM	128	100%	67850	12	79.8%	26.6%	98.5%
RNN	128	100%	17930	10	71.3%	24.5%	89%
LSNN	100(A), 120(R)	12%	8185 (full 68210)	12	94.2%	0.3%	94.7%
LSNN	100(A), 200(R)	12%	14041 (full 117010)	1	-	-	95.7%
LSNN	350(A), 350(R)	12%	66360 (full 553000)	1	-	-	96.1%
LSNN	100(A), 120(R)	100%	68210	10	92.0%	0.7%	93.3%
LIF	220	100%	68210	10	60.9%	2.7%	63.3%

Table C.1: Results on 1 ms per pixel sequential MNIST task.

Model	# neurons	conn.	# params	# runs	mean	std.	max.
LSTM	128	100%	67850	12	48.2%	39.9%	98.0%
RNN	128	100%	17930	12	30%	23.6%	67.9%
LSNN	100(A), 120(R)	12%	8185 (full 68210)	12	93.8%	5.8%	96.4%
LSNN	350(A), 350(R)	12%	66360 (full 553000)	1	-	-	97.1%
LSNN	100(A), 120(R)	100%	68210	10	90.5%	1.4%	93.7%
LIF	220	100%	68210	11	34.6%	8.8%	51.8%

Table C.2: Results on 2 ms per pixel sequential MNIST task.

TIMIT setup: To investigate if the performance of LSNNs can scale to real world problems, we considered the TIMIT speech recognition task. We focused on the frame-wise classification where the LSNN has to classify each audio-frame to one of the 61 phoneme classes.

We followed the convention of Halberstadt (Glass et al., 1999) for grouping of training, validation, and testing sets (3696, 400, and 192 sequences respectively). The performance was evaluated on the *core test set* for consistency with the literature. Raw audio is preprocessed into 13 Mel Frequency Cepstral Coefficients (MFCCs) with frame size 10 ms and on input window of 25 ms. We computed the first and the second order derivatives of MFCCs and combined them, resulting in 39 input channels. These 39 input channels were mapped to 39 input neurons which unlike in MNIST emit continuous values $x_i(t)$ instead of spikes, and these values were directly used in equation C.3 for the currents of the postsynaptic neurons.

Since we simulated the LSNN network in 1 ms time steps, every input frame which represents 10 ms of the input audio signal was fed to the LSNN network for 10 consecutive 1 ms steps. The softmax output of the LSNN was averaged over every 10 steps to produce the prediction of the phone in the current input frame. The LSNN was rewired with global connectivity level of 20%.

Parameter values: For adaptive neurons, we used $\beta_j = 1.8$, and for regular spiking neurons we used $\beta_j = 0$ (i.e. B_j is constant). The baseline threshold voltage was $b_j^0 = 0.01$ and the membrane time constant $\tau_m = 20$ ms. Networks were trained using the default Adam optimizer, and a learning rate initialized at 0.01. The dampening factor for training was $\gamma = 0.3$.

For sequential MNIST, all networks were trained for 36000 iterations with a batch size of 256. Learning rate was decayed by a factor 0.8 every 2500 iterations. The adaptive neurons in the LSNN had an adaptation time constant $\tau_a = 700$ ms (1400 ms) for 1 ms (2 ms) per pixel setup. The baseline artificial RNN contained 128 hidden units with the hyperbolic tangent activation function. The LIF network was formed by a fully connected population of 220 regular spiking neurons.

For TIMIT, the LSNN network consisted of 300 regular neurons and 100 adaptive neurons which resulted in approximately 400000 parameters. Network was trained for 80 epochs with batches of 32 sequences. Adaptation time constant of adaptive neurons was set to $\tau_a = 200$ ms. Refractory period of the neurons was set to 2 ms, the membrane time constant of the output Y neurons to 3 ms, and the synaptic delay was randomly picked from $\{1, 2\}$ ms.

We note that due to the rewiring of the LSNN using DEEP R Bellec et al., 2018a method, only a small fraction of the weights had non-zero values (8185 in MNIST, ~ 80000 in TIMIT).

C.4 LSNNs learn-to-learn from a teacher

Experimental setup:

Function families: The LSNN was trained to implement a regression algorithm on a family of functions $\mathcal{F}$. Two specific families were considered: In the first function family, the functions were defined by feed-forward neural networks with 2 inputs, 1 hidden layer consisting of 10 hidden neurons, and 1 output, where all the parameters (weights and biases) were chosen uniformly randomly between $[-1, 1]$. The inputs were between $[-1, 1]$ and the outputs were scaled to be between $[0, 1]$. We call these networks Target Networks (TNs). In the second function family, 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.

LSNN setup and training schedule: The standard LSNN model was used, with 300 hidden neurons for the TN family of learning tasks, and 100 for the sinusoidal family. Of these, 40% were adaptive in all simulations. We used all-to-all connectivity between all neurons (regular and adaptive). 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 the step consists of.

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 (depending on whether its a TN or sinusoidal). 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 vector $\mathbf{x}$ with its dimensionality d and range determined by the target function being used ($d = 2$ for TNs, $d = 1$ for sinusoidal target function). In addition, at each step, the LSNN also got the target value $C(\mathbf{x}')$ 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, $C(\mathbf{x}')$ is set to $\mathbf{0}$).

All the weights of the LSNN were updated using our variant of BPTT, once per **iteration**, where an **iteration** consists of a batch of 10 **episodes**, and the weight updates are accumulated across episodes in an iteration. The Adam Kingma and Ba, 2014 variant of BP 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. Specifically, the regularization term R is defined as the mean squared difference between the average neuron firing rate in the LSNN and a target of 20 Hz. The total loss L was then given by $L = MSE + 30\,R$. In this way, we induce the LSNN to use sparse firing. We trained the LSNN for 5000 iterations in all cases.

Parameter values: The LSNN parameters were as follows: 5 ms neuronal refractory period, delays spread uniformly between $0 - 5$ ms, adaptation time constants of the adaptive neurons spread uniformly between $1 - 1000$ ms, $\beta = 1.6$ 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 $\gamma = 0.4$.

Analysis and comparison: 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. 2C, for each episode consisting of 500 steps, the mean spiking trace from a random subset of 450 steps was used to train the linear regressor, and the mean spiking trace from remaining 50 steps was used to calculate the test error. The reported baseline is the mean of the test error over one batch of 10 episodes with error bars of one standard deviation. In Fig. 2E, for each episode, after every step k, the mean spiking traces from the first $k - 1$ steps were used to train the linear regressor, and the test error was calculated using the mean spiking trace for the kth step. The reported baseline is a mean of the test error over one batch of 10 episodes with error bars of one standard deviation.

For the case where neural networks defined the function family, the total test MSE was 0.0056 ± 0.0039 (linear baseline MSE was 0.0217 ± 0.0046). For the sinusoidal function family, the total test MSE was 0.3134 ± 0.2293 (linear baseline MSE was 1.4592 ± 1.2958).

Comparison with backprop: The comparison was done for the case where the LSNN is trained on the function family defined by target networks. A feed-forward (FF)

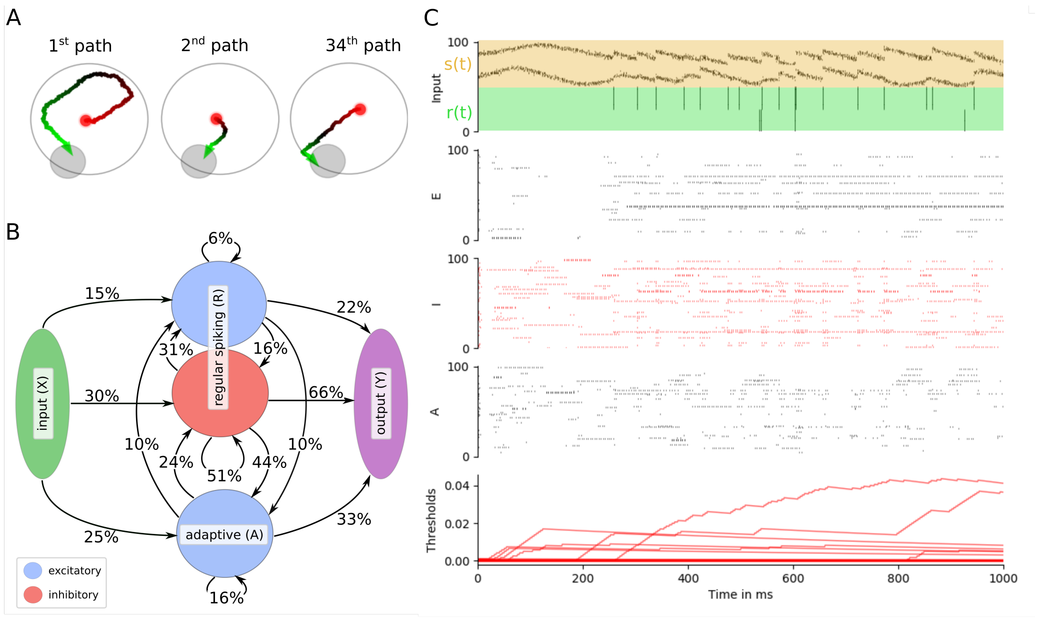


Fig. C.1: Meta-RL results for an LSNN. A Samples of paths after training. **B** Connectivity between sub-populations of the network after training. The global connectivity in the network was constrained to 20%. **C** The network dynamics that produced the behavior shown in A. Raster plots and thresholds are displayed as in Fig. 1.D, only 1 second and 100 neurons are shown in each raster plots.

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 LSNN. Therefore the FF had 2 inputs, one for each of x_1 and x_2. The error reported in Fig 2F is the mean training error over 10 batches 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.

C.5 LSNNs learn-to-learn from reward

Experimental setup:

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 centres 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. C.1C). 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. C.1C). All neuron in the first group spike in synchrony each time a reward of 1 is received (i.e., the goal was reached), and the second group spikes 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 $\mathbf{a}(t) = (a_x(t), a_y(t))^T$ 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 $a_{scale}\,\mathbf{a}(t)$. 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.

Network training: 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:

$$\mathcal{L}(\boldsymbol{\theta}) \quad = \quad -\frac{1}{KT}\sum_{k<K}\sum_{t<T} O^{PPO}(\boldsymbol{\theta}_{old}, \boldsymbol{\theta}, t, k) + \mu_v \left(R(t,k) - V_{\boldsymbol{\theta}}(t,k)\right)^2 \tag{C.8}$$

$$-\mu_e H(\pi_{\boldsymbol{\theta}}(k,t)) + \mu_{firing}\frac{1}{n}\sum_j ||\frac{1}{KT}\sum_{k,t} z_j(t,k) - f^0||^2, \tag{C.9}$$

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.

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 D

Methods for "Biologically inspired alternatives to BPTT for learning in recurrent neural nets"

D.1 Model definition and mathematical basis for *e-prop*

D.1.1 General network model

Our proposed learning algorithms for recurrent neural networks can be applied to a large class of spiking and non-spiking neural network models. We assume that the state at time t of each neuron j in the network can be described by an internal state vector $\boldsymbol{s}_j^t \in \mathbb{R}^d$ and an observable state z_j^t. The internal state includes internal variables of the neuron such as its activation or membrane potential. The observable state is given by the output of the neuron (analog output for ANNs and spiking output for SNNs). The dynamics of a neuron's discrete-time state evolution is described by two functions $M(\boldsymbol{s}, \boldsymbol{z}, \boldsymbol{x}, \boldsymbol{\theta})$ and $f(\boldsymbol{s})$, where $\boldsymbol{s}$ is an internal state vector, $\boldsymbol{z}$ is a vector of the observable network state (i.e., outputs of all neurons in the network), $\boldsymbol{x}$ is the vector of inputs to the network, and $\boldsymbol{\theta}$ denotes the vector of network parameters (i.e., synaptic weights). In particular, for each neuron j the function M maps from the current network state observable to that neuron to its next internal state, and f maps from its internal state to its observable state (neuron output):

$$\boldsymbol{s}_j^t = M(\boldsymbol{s}_j^{t-1}, \boldsymbol{z}^{t-1}, \boldsymbol{x}^t, \boldsymbol{\theta}), \tag{D.1}$$

$$z_j^t = f(\boldsymbol{s}_j^t), \tag{D.2}$$

where $\boldsymbol{z}^t$ ($\boldsymbol{x}^t$) denotes the vector of observable states of all network (input) neurons at time t. A third function E defines the error of the network within some time interval $0, \dots, T$. It is assumed to depend only on the observable states $E(\boldsymbol{z}^1, \dots, \boldsymbol{z}^T)$.

We explicitly distinguish betweens partial derivative and total derivatives in our notation. We write $\frac{\partial M}{\partial s}(\boldsymbol{s}^*, \boldsymbol{z}^*, \boldsymbol{x}^*, \boldsymbol{\theta})$ to denote the partial derivative of the function M with respect to $\boldsymbol{s}$, applied to particular arguments $\boldsymbol{s}^*, \boldsymbol{z}^*, \boldsymbol{x}^*, \boldsymbol{\theta}$. To simplify notation, we define the shortcuts $\frac{\partial \boldsymbol{s}_j^t}{\partial \boldsymbol{s}_j^{t-1}} \overset{\text{def}}{=} \frac{\partial M}{\partial s}(\boldsymbol{s}_j^{t-1}, \boldsymbol{z}^{t-1}, \boldsymbol{x}^t, \boldsymbol{\theta})$, $\frac{\partial \boldsymbol{s}_j^t}{\partial \theta_{ji}} \overset{\text{def}}{=} \frac{\partial M}{\partial \theta_{ji}}(\boldsymbol{s}_j^{t-1}, \boldsymbol{z}^{t-1}, \boldsymbol{x}^t, \boldsymbol{\theta})$, and $\frac{\partial z_j^t}{\partial \boldsymbol{s}_j^t} \overset{\text{def}}{=} \frac{\partial f}{\partial s}(\boldsymbol{s}_j^t)$.

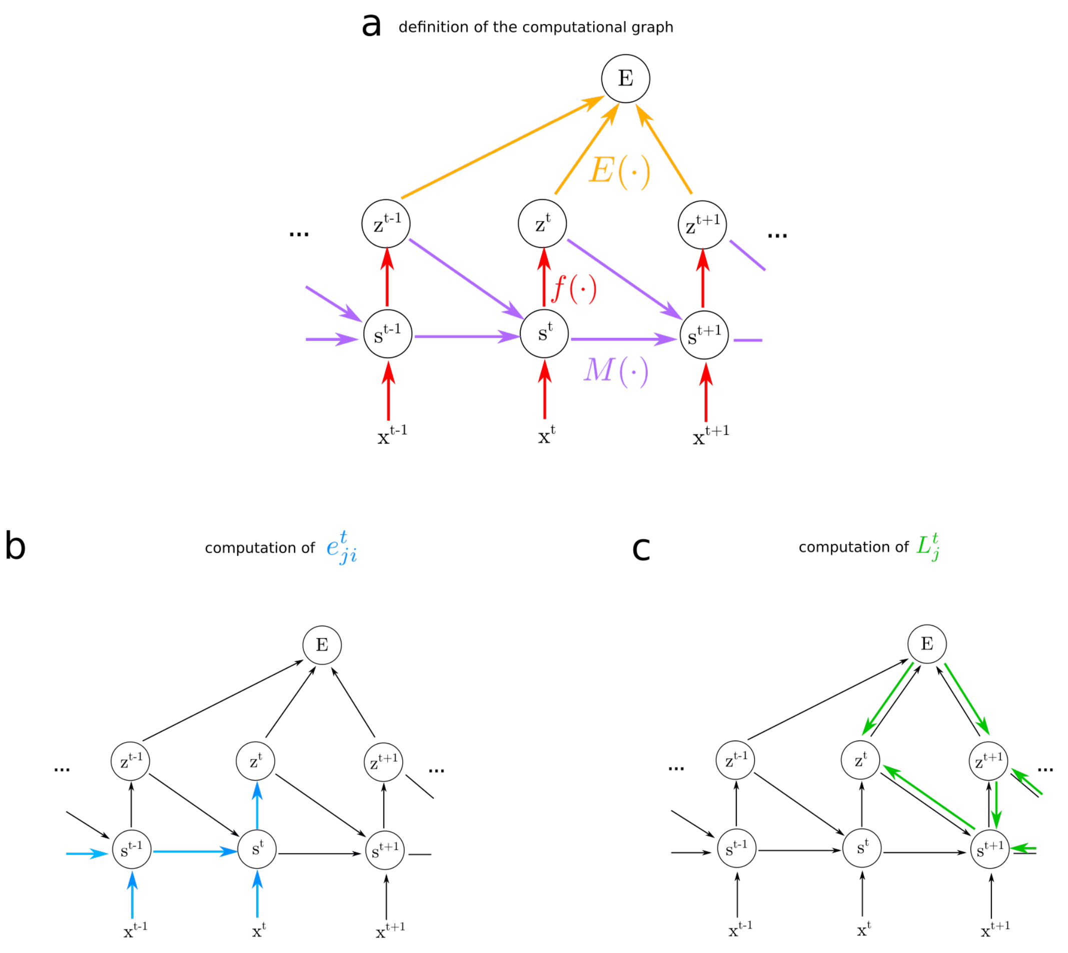


Fig. D.1: Computational graph a) Assumed mathematical dependencies between neuron states $\boldsymbol{s}_j^t$, neuron outputs $\boldsymbol{z}^t$, network inputs $\boldsymbol{x}^t$, and the network error E through the mathematical functions $f(\cdot)$, $M(\cdot)$ and $E(\cdot)$ represented by coloured arrows. **b)** The dependencies involved in the computation of the eligibility traces e_{ji}^t are shown in blue. **c)** The dependencies involved in the computation of the learning signal L_j^t are shown in green.

To emphasize that $\frac{\partial \boldsymbol{s}_j^t}{\partial \boldsymbol{s}_j^{t-1}}$ is a matrix of shape $d \times d$, and because it has an important role in the following derivation and in definition of eligibility traces, we also use the further notation $D_j^t = \frac{\partial \boldsymbol{s}_j^{t+1}}{\partial \boldsymbol{s}_j^t}$. Note that we write gradients as row vectors and states as column vectors.

D.1.2 Proof of factorization (equation (4.1))

We provide here the proof for equation (4.1), i.e., we show that the total derivative of the error function E with respect to the parameters $\boldsymbol{\theta}$ can be written as a product of learning signals L_j^t and eligibility traces e_{ji}^t. First, recall that in BPTT the error gradient is decomposed as (see equation (12) in Werbos, 1990):

$$\frac{dE}{d\theta_{ji}} = \sum_t \frac{dE}{d\mathbf{s}_j^t} \cdot \frac{\partial \mathbf{s}_j^t}{\partial \theta_{ji}} \,, \tag{D.3}$$

where $\frac{dE}{d\mathbf{s}_j^t}$ is the total derivative of the error E with respect to the neuron states $\mathbf{s}_j^t$ at time step t. $\frac{dE}{d\mathbf{s}_j^t}$ can be expressed recursively as a function of the same derivative at the next time step $\frac{dE}{d\mathbf{s}_j^{t+1}}$ by applying the chain rule at the node $\mathbf{s}_j^t$ of the computational graph shown in Figure D.1c:

$$\begin{aligned} \frac{dE}{d\mathbf{s}_j^t} &= \frac{dE}{dz_j^t}\frac{\partial z_j^t}{\partial \mathbf{s}_j^t} + \frac{dE}{d\mathbf{s}_j^{t+1}}\frac{\partial \mathbf{s}_j^{t+1}}{\partial \mathbf{s}_j^t} && \text{(D.4)} \\ &= L_j^t \frac{\partial z_j^t}{\partial \mathbf{s}_j^t} + \frac{dE}{d\mathbf{s}_j^{t+1}} D_j^t, && \text{(D.5)} \end{aligned}$$

where we defined the *learning signal* for neuron j at time t as $L_j^t \overset{\text{def}}{=} \frac{dE}{dz_j^t}$. The resulting recursive expansion ends at the last time step T, i.e., $\frac{dE}{d\mathbf{s}_j^{T+1}} = 0$. If one substitutes the recursive formula (D.5) into the definition of the error gradients (D.3), one gets:

$$\begin{aligned} \frac{dE}{d\theta_{ji}} &= \sum_t \left(L_j^t \frac{\partial z_j^t}{\partial \mathbf{s}_j^t} + \frac{dE}{d\mathbf{s}_j^{t+1}} D_j^t \right) \cdot \frac{\partial \mathbf{s}_j^t}{\partial \theta_{ji}} && \text{(D.6)} \\ &= \sum_t \left(L_j^t \frac{\partial z_j^t}{\partial \mathbf{s}_j^t} + \big(L_j^{t+1} \frac{\partial z_j^{t+1}}{\partial \mathbf{s}_j^{t+1}} + (\cdots) D_j^{t+1} \big) D_j^t \right) \cdot \frac{\partial \mathbf{s}_j^t}{\partial \theta_{ji}} \,. && \text{(D.7)} \end{aligned}$$

The following equation is the main equation for understanding the transformation from BPTT into *e-prop*. The key idea is to collect all terms which are multiplied with the learning signal $L_j^{t'}$ at a given time t'. These are only terms that concern events in the computation of neuron j at time t', and these do not depend on future errors or variables. Hence one can collect them conceptually into an internal eligibility trace e_{ji}^t for neuron j which can be computed autonomously within neuron j in an online manner.

To this end, we write the term in parentheses in equation (D.7) into a second sum indexed by t' and exchange the summation indices to pull out the learning signal L_j^t. This expresses the error gradient as a sum of learning signals L_j^t multiplied by some factor indexed by ji, which implicitly defines what we call eligibility traces and eligibility vectors:

$$\frac{dE}{d\theta_{ji}} = \sum_t \sum_{t' \geq t} L_j^{t'} \frac{\partial z_j^{t'}}{\partial \mathbf{s}_j^{t'}} D_j^{t'-1} \cdots D_j^t \cdot \frac{\partial \boldsymbol{s}_j^t}{\partial \theta_{ji}} \tag{D.8}$$

$$= \sum_{t'} L_j^{t'} \frac{\partial z_j^{t'}}{\partial \mathbf{s}_j^{t'}} \underbrace{\sum_{t \leq t'} D_j^{t'-1} \cdots D_j^t \cdot \frac{\partial \boldsymbol{s}_j^t}{\partial \theta_{ji}}}_{\overset{\text{def}}{=} \boldsymbol{\epsilon}_{ji}^{t'}} . \tag{D.9}$$

Here, we use the identity matrix for the $D_j^{t-1} \cdots D_j^t$ where $t' - 1 < t$. Finally, seeing that the eligibility vectors $\boldsymbol{\epsilon}_{ji}^{t'}$ can also be computed recursively as in equation (4.2), it proves the equation (4.1), given the definition of eligibility traces and learning signals in (4.3) and (4.4).

D.1.3 Leaky integrate-and-fire neuron model

We define here the leaky integrate-and-fire (LIF) spiking neuron model, and exhibit the update rules that result from *e-prop* for this model. We consider LIF neurons simulated in discrete time. In this case the internal state $\boldsymbol{s}_j^t$ is one dimensional and contains only the membrane voltage v_j^t. The observable state $z_j^t \in \{0, 1\}$ is binary, indicating a spike ($z_j^t = 1$) or no spike ($z_j^t = 0$) at time t. The dynamics of the LIF model is defined by the equations:

$$v_j^{t+1} = \alpha v_j^t + \sum_{i \neq j} \theta_{ji}^{\text{rec}} z_i^t + \sum_i \theta_{ji}^{\text{in}} x_i^t - z_j^t v_{\text{th}} \tag{D.10}$$

$$z_j^t = H\left(\frac{v_j^t - v_{\text{th}}}{v_{\text{th}}}\right), \tag{D.11}$$

where $x_i^t = 1$ indicates a spike from the input neuron i at time step t ($x_i^t = 0$ otherwise), θ_{ji}^{rec} (θ_{ji}^{in}) is the synaptic weight from network (input) neuron i to neuron j, and H denotes the Heaviside step function. The decay factor α is given by $e^{-\delta t/\tau_m}$, where δt is the discrete time step (1 ms in our simulations) and τ_m is the membrane time constant. Due to the term $-z_j^t v_{\text{th}}$ in equation (D.10), the neurons membrane voltage is reset to a lower value after an output spike.

Eligibility traces and error gradients: Considering the LIF model defined above, we derive the resulting eligibility traces and error gradients. By definition of the model in equation (D.10), we have $D_j^t = \frac{\partial v_j^{t+1}}{\partial v_j^t} = \alpha$ and $\frac{\partial v_j^t}{\partial \theta_{ji}} = z_i^{t-1}$. Therefore, using the definition of eligibility vectors in equation (4.2), one obtains a simple geometric series and one can write:

$$\epsilon_{ji}^{t+1} = \sum_{t' \leq t} \alpha^{t-t'} z_i^{t'} \overset{\text{def}}{=} \hat{z}_i^t , \tag{D.12}$$

and the eligibility traces are written:

$$e_{ji}^{t+1} = h_j^t \hat{z}_i^t \, . \tag{D.13}$$

In other words, the eligibility vector is one dimensional and depends only on the presynaptic neuron i and in fact, corresponds to the filtered presynaptic spike train. To exhibit resulting the eligibility trace defined by equation (4.3), this requires to compute the derivative $\frac{\partial z_j^t}{\partial v_j^t}$, which is ill-defined in the case of LIF neurons because it requires the derivative of the discontinuous function H. As shown in (Bellec et al., 2018c), we can alleviate this by using a pseudo-derivative h_j^t in place of $\frac{\partial z_j^t}{\partial v_j^t}$, given by $h_j^t = \gamma \max\left(0, 1 - |\frac{v_j^t - v_{\text{th}}}{v_{\text{th}}}|\right)$ where $\gamma = 0.3$ is a constant called dampening factor. The gradient of the error with respect to a recurrent weight θ_{ji}^{rec} thus takes on the following form, reminiscent of spike-timing dependent plasticity:

$$\frac{dE}{d\theta_{ji}^{\text{rec}}} = \sum_t \frac{dE}{dz_j^t} h_j^t \hat{z}_i^{t-1} \, . \tag{D.14}$$

A similar derivation leads to the gradient of the input weights. In fact, one just needs to substitute recurrent spikes z_i^t by input spikes x_i^t. The gradient with respect to the output weights does not depend on the neuron model and is therefore detailed another paragraph (see equation (D.38)).

Until now refractory periods were not modeled to simplify the derivation. To introduce a simple model of refractory periods that is compliant with the theory, one can further assume that z_j^t and $\frac{\partial z_j}{\partial \boldsymbol{s}_j}$ are fixed to 0 for a short refractory period after each spike of neuron j. Outside of the refractory period the neuron dynamics are otherwise unchanged.

D.1.4 Leaky integrate-and-fire neurons with threshold adaptation

We derive the learning rule defined by *e-prop* for a LIF neuron model with an adaptive threshold. For this model, the internal state is given by a two-dimensional vector $\boldsymbol{s}_j^t := (v_j^t, a_j^t)^T$, where v_j^t denotes the membrane voltage as in the LIF model, and a_j^t is a threshold adaptation variable. As for the LIF model above, the voltage dynamics is defined by equation (D.10). The spiking threshold A_j^t at time t is given by

$$A_j^t = v_{\text{th}} + \beta a_j^t \, , \tag{D.15}$$

where v_{th} denotes the baseline-threshold. Output spikes are generated when the membrane voltage crosses the adaptive threshold $z_j^t = H\left(\frac{v_j^t - A_j^t}{v_{\text{th}}}\right)$, and the threshold adaptation evolves according to

$$a_j^{t+1} = \rho a_j^t + H\left(\frac{v_j^t - A_j^t}{v_{\text{th}}}\right). \tag{D.16}$$

The decay factor ρ is given by $e^{-\delta t/\tau_a}$, where δt is the discrete time step (1 ms in our simulations) and τ_a is the adaptation time constant. In other words, the neuron's threshold is increased with every output spike and decreases exponentially back to the baseline threshold.

Eligibility traces: Because of the extended state, we obtain one two-dimensional eligibility vector per synaptic weight: $\epsilon_{ji}^t := (\epsilon_{ji,v}^t, \epsilon_{ji,a}^t)^T$ and the matrix D_j^t is a 2×2 matrix. On its diagonal one finds the terms $\frac{\partial v_j^{t+1}}{\partial v_j^t} = \alpha$ and $\frac{\partial a_j^{t+1}}{\partial a_j^t} = \rho - h_j^t \beta$.

Above and below the diagonal, one finds respectively $\frac{\partial v_j^{t+1}}{\partial a_j^t} = 0, \frac{\partial a_j^{t+1}}{\partial v_j^t} = h_j^t$. One can finally compute the eligibility traces using its definition in equation (4.3). The component of the eligibility vector associated with the voltage remains the same as in the LIF case and only depends on the presynaptic neuron: $\epsilon_{ji,v}^t = \hat{z}_i^{t-1}$. For the component associated with the adaptive threshold we find the following recursive update:

$$\epsilon_{ji,a}^{t+1} = h_j^t \hat{z}_i^{t-1} + (\rho - h_j^t \beta)\epsilon_{ji,a}^t \,, \tag{D.17}$$

and this results in an eligibility trace of the form:

$$e_{ji}^{t+1} = h_j^t \left(\hat{z}_i^{t-1} - \beta \epsilon_{ji,a}^t\right). \tag{D.18}$$

This results in the following equation for the gradient of recurrent weights:

$$\frac{dE}{d\theta_{ji}^{\text{rec}}} = \sum_t \frac{dE}{dz_j^t} h_j^t \left(\hat{z}_i^{t-1} - \beta \epsilon_{ji,a}^t\right) . \tag{D.19}$$

The eligibility trace for input weights is again obtained by replacing recurrent spikes z_i^t with input spikes x_i^t.

D.1.5 Artificial neuron models

The dynamics of recurrent artificial neural networks is usually given by $s_j^t = \alpha s_j^{t-1} + \sum_i \theta_{ji}^{\text{rec}} z_i^{t-1} + \sum_i \theta_{ji}^{\text{in}} x_j^t$ with $z_j^t = \sigma(s_j^t)$, where $\sigma : \mathbb{R} \to \mathbb{R}$ is some activation function (often a sigmoidal function in RNNs). We call the first term the leak term in analogy with LIF models. For $\alpha = 0$ this term disappears, leading to the arguably most basic RNN model. If $\alpha = 1$, it models a recurrent version of residual networks.

Eligibility traces: For such model, one finds that $D_j^t = \alpha$ and the eligibility traces are equal to $e_{ji}^t = h_j^t \hat{z}_i^{t-1}$ with $\hat{z}_i^t \overset{\mathrm{def}}{=} \sum_{t' \leq t} z_i^{t'} \alpha^{t-t'}$. The resulting *e-prop* update is written as follows (with σ' the derivative of the activation function):

$$\frac{dE}{d\theta_{ji}^{\mathrm{rec}}} = \sum_t \frac{dE}{dz_j^t} \sigma'(s_j^t) \hat{z}_i^{t-1}. \tag{D.20}$$

Although simple, this derivation provides insight in the relation between BPTT and *e-prop*. If the neuron model does not have neuron specific dynamics $\alpha = 0$, the factorization of *e-prop* is obsolete in the sense that the eligibility trace does not propagate any information from a time step to the next $D_j^t = 0$. Thus, one sees that *e-prop* is most beneficial for models with rich internal neural dynamics.

D.1.6 LSTM

For LSTM units, (Hochreiter and Schmidhuber, 1997) the internal state of the unit is the content of the memory cell and is denoted by c_j^t, the observable state is denoted by h_j^t. One defines the network dynamics that involves the usual input, forget and output gates (denoted by $\mathscr{i}_j^t$, $\mathscr{f}_j^t$, and $\mathscr{o}_j^t$) and the cell state candidate $\widetilde{c}_j^t$ as follows (we ignore biases for simplicity):

$$\begin{aligned}
\mathscr{i}_j^t &= \sigma\big(\sum_i \theta_{ji}^{\mathrm{rec},\mathscr{i}} h_i^{t-1} + \sum_i \theta_{ji}^{\mathrm{in},\mathscr{i}} x_i^t\big) && \text{(D.21)}\\
\mathscr{f}_j^t &= \sigma\big(\sum_i \theta_{ji}^{\mathrm{rec},\mathscr{f}} h_i^{t-1} + \sum_i \theta_{ji}^{\mathrm{in},\mathscr{f}} x_i^t\big) && \text{(D.22)}\\
\mathscr{o}_j^t &= \sigma\big(\sum_i \theta_{ji}^{\mathrm{rec},\mathscr{o}} h_i^{t-1} + \sum_i \theta_{ji}^{\mathrm{in},\mathscr{o}} x_i^t\big) && \text{(D.23)}\\
\widetilde{c}_j^t &= \tanh\big(\sum_i \theta_{ji}^{\mathrm{rec},c} h_i^{t-1} + \sum_i \theta_{ji}^{\mathrm{in},c} x_i^t\big). && \text{(D.24)}
\end{aligned}$$

Using those intermediate variables as notation short-cuts, one can now write the update of the states of LSTM units in a form that we can relate to *e-prop*:

$$\begin{aligned}
c_j^t &= M(c_j^{t-1}, \boldsymbol{h}^{t-1}, \boldsymbol{\theta}) = \mathscr{f}_j^t c_j^{t-1} + \mathscr{i}_j^t \widetilde{c}_j^t && \text{(D.25)}\\
h_j^t &= f(c_j^t, \boldsymbol{h}^{t-1}, \boldsymbol{\theta}) = \mathscr{o}_j^t c_j^t. && \text{(D.26)}
\end{aligned}$$

Eligibility traces: There is one difference between LSTMs and the previous neuron models used for *e-prop*: the function f depends now on the previous observable state $\boldsymbol{h}^t$ and the parameters through the output gate $\mathscr{o}_j^t$. However the derivation of the gradients $\frac{dE}{d\theta_{ji}}$ in the paragraph "Proof of factorization" is still valid for deriving the gradients with respect to the parameters of the input gate, forget gate and cell state candidate. For these parameters we apply the general theory as follows. We compute $D_j^t = \frac{\partial c_j^{t+1}}{\partial c_j^t} = \mathscr{f}_j^t$ and for each variable $\theta_{ji}^{A,B}$ with A being either "in" or "rec" and B being i, f, or c, we compute a set of eligibility traces. If we take the example the recurrent weights for the input gate $\theta_{ji}^{\text{rec},i}$, the eligibility vectors are updated according to:

$$\epsilon_{ji}^{\text{rec},i,t} = f_j^{t-1} \epsilon_{ji}^{\text{rec},i,t-1} + \tilde{c}_j^t i_j^t (1 - i_j^t) h_i^t \,, \tag{D.27}$$

the eligibility traces are then written:

$$e_{ji}^{\text{rec},i,t} = o_j^t \epsilon_{ji}^{\text{rec},i,t} , \tag{D.28}$$

and the gradients are of the form

$$\frac{dE}{d\theta_{ji}^{\text{rec},i}} = \sum_t \frac{dE}{dh_j^t} o_j^t \epsilon_{ji}^{\text{rec},i,t} \,. \tag{D.29}$$

For the parameters $\theta_{ji}^{\text{rec},o}$ of the output gate which take part in the function f, we need to derive the gradients in a different manner. As we still assume that $E(\boldsymbol{h}^1, \ldots, \boldsymbol{h}^T)$ depends on the observable state only, we can follow the derivation of BPTT with a formula analogous to (D.3). This results in a gradient expression involving local terms and the same learning signal as used for other parameters. Writing $\frac{\partial f}{\partial \theta_{ji}^{\text{rec},o}}$ as $\frac{\partial h_j^t}{\partial \theta_{ji}^{\text{rec},o}}$ the gradient update for the output gate takes the form:

$$\frac{dE}{d\theta_{ji}^{\text{rec},o}} = \sum_t \frac{dE}{dh_j^t} \frac{\partial h_j^t}{\partial \theta_{ji}^{\text{rec},o}} = \sum_t \frac{dE}{dh_j^t} c_j^t o_j^t (1 - o_j^t) h_i^{t-1}. \tag{D.30}$$

D.2 *E-prop 1*

D.2.1 Formalization of *E-prop 1*

E-prop 1 follows the general *e-prop* framework and applies to all the models above. Its specificity is the choice of learning signal. In this first variant, we make two approximations: future errors are ignored so that one can compute the learning signal in real-time, and learning signals are fed back with random connections. The specific realizations of the two approximations are discussed independently in the following and implementation details are provided.

Ignoring future errors: The first approximation is to focus on the error at the present time t and ignore dependencies on future errors in the computation of the total derivative $\frac{dE}{dz_j^t}$. Using the chain rule, this total derivative expands as $\frac{dE}{dz_j^t} = \frac{\partial E}{\partial z_j^t} + \frac{dE}{d\boldsymbol{s}_j^{t+1}} \frac{\partial \boldsymbol{s}_j^{t+1}}{\partial z_j^t}$, and neglecting future errors means that we ignore the second term of this sum. As a result the total derivative $\frac{dE}{dz_j^t}$ is replaced by the partial derivative $\frac{\partial E}{\partial z_j^t}$ in equation (4.4).

Synaptic weight updates under *e-prop 1*: Usually, the output of an RNN is given by the output of a set of readout neurons which receive input from network neurons, weighted by synaptic weights θ_{kj}^{out}. In the case of an RSNN, in order to be able to generate non-spiking outputs, readouts are modeled as leaky artificial neurons. More precisely, the output of readout k at time t is given by

$$y_k^t = \kappa y_k^{t-1} + \sum_j \theta_{kj}^{\text{out}} z_j^t + b_k^{\text{out}}, \tag{D.31}$$

where $\kappa \in [0, 1]$ defines the leak and b_k^{out} denotes the readout bias. The leak factor κ is given by $e^{-\delta t/\tau_{out}}$, where δt is the discrete time step and τ_{out} is the membrane time constant). In the following derivation of weight updates under *e-prop 1*, we assume such readout neurons. Additionally, we assume that the error function is given by the mean squared error $E = \frac{1}{2}\sum_{t,k}(y_k^t - y_k^{*,t})^2$ with $y_k^{*,t}$ being the target output at time t (see the following paragraph on "classification" when the cross entropy error is considered).

In this case, the partial derivative $\frac{\partial E}{\partial z_i^t}$ has the form:

$$\frac{\partial E}{\partial z_j^t} = \theta_{kj}^{\text{out}} \sum_{t' \geq t} (y_k^{t'} - y_k^{*,t'}) \kappa^{t'-t}. \tag{D.32}$$

This seemingly poses a problem for a biologically plausible learning rule, because the partial derivative is a weighted sum over the future. This issue can however easily be solved as we show below. Using equation (4.1) for the particular case of *e-prop 1*, we insert $\frac{\partial E}{\partial z_j^t}$ in-place of the total derivative $\frac{dE}{dz_j^t}$ which leads to an estimation $\widehat{\frac{dE}{d\theta_{ji}}}$ of the true gradient given by:

$$\begin{aligned}
\widehat{\frac{dE}{d\theta_{ji}}} &= \sum_t \frac{\partial E}{\partial z_j^t} e_{ji}^t && \text{(D.33)}\\
&= \sum_{k,t} \theta_{kj}^{\text{out}} \sum_{t' \geq t} (y_k^t - y_k^{*,t}) \kappa^{t'-t} e_{ji}^t && \text{(D.34)}\\
&= \sum_{k,t'} \theta_{kj}^{\text{out}} (y_k^{t'} - y_k^{*,t'}) \sum_{t \leq t'} \kappa^{t'-t} e_{ji}^t \,, && \text{(D.35)}
\end{aligned}$$

where we inverted sum indices in the last line. The second sum indexed by t is now over previous events that can be computed in real time, it computes a filtered copy of the eligibility trace e_{ji}^t. With this additional filtering of the eligibility trace with a time constant equal to that of the temporal averaging of the readout neuron, we see that *e-prop 1* takes into account the latency between an event at time t' and its impact of later errors at time t within the averaging time window of the readout. However, note that this time constant is a few tens of milliseconds in our experiments which is negligible in comparison to the decay of the eligibility traces of adaptive neurons which are one to two orders of magnitude larger (see Figure 4.2).

Equation (D.35) holds for any neuron model. In the case of LIF neurons, the eligibility traces are given by equation (D.13), and one obtains the final expression of the error gradients after substituting these expressions in (D.35). Implementing weight updates with gradient descent and learning rate η, the updates of the recurrent weights are given by

$$\Delta\theta_{ji}^{\mathrm{rec}} = \eta \sum_t \Big(\sum_k \theta_{kj}^{\mathrm{out}} (y_k^{*,t} - y_k^t) \Big) \sum_{t' \leq t} \kappa^{t-t'} h_j^{t'} \hat{z}_i^{t'-1} \, . \qquad \text{(D.36)}$$

When the neurons are equipped with adaptive thresholds as in LSNNs, one replaces the eligibility traces with their corresponding definitions. It results that an additional term $\epsilon_{ji,a}^t$ as defined in equation (D.17) is introduced in the weight update:

$$\Delta\theta_{ji}^{\mathrm{rec}} = \eta \sum_t \Big(\sum_k \theta_{kj}^{\mathrm{out}} (y_k^{*,t} - y_k^t) \Big) \sum_{t' \leq t} \kappa^{t-t'} h_j^{t'} \left(\hat{z}_i^{t'-1} - \beta \epsilon_{ji,a}^{t'} \right) \, . \qquad \text{(D.37)}$$

Both equations (D.36) and (D.37) can be derived similarly for the input weights $\theta_{ji}^{\mathrm{in}}$, and it results in the same learning rule with the only difference that $\hat{z}_i^{t'-1}$ is replaced by a trace of the spikes x_i^t of input neuron i. For the output connections the gradient $\frac{dE}{d\theta_{kj}^{\mathrm{out}}}$ can be derived as for isolated linear readout neurons and it does not need to rely on the theory of *e-prop*. The resulting weight update is:

$$\Delta\theta_{kj}^{\mathrm{out}} = \eta \sum_t (y_k^{*,t} - y_k^t) \sum_{t' \leq t} \kappa^{t-t'} z_j^{t'} \, . \qquad \text{(D.38)}$$

Random feed-back matrices: According to equations (D.36) and (D.37), the signed error signal from readout k communicated to neuron j has to be weighted with $\theta_{kj}^{\mathrm{out}}$. That is, the synaptic efficacies of the feedback synapses have to equal those of the feed-forward synapses. This general property of backpropagation-based algorithms is a problematic assumption for biological circuits. It has been shown however in (Samadi et al., 2017; Nøkland, 2016) that for many tasks, an approximation where the feedback weights are chosen randomly works well. We adopt this approximation in *e-prop 1*. Therefore we replace in equations (D.36) and (D.37) the weights $\theta_{kj}^{\mathrm{out}}$ by fixed random values B_{jk}^{random}. For a LIF neuron the learning rule (D.36) becomes with random feedback weights:

$$\Delta\theta_{ji}^{\mathrm{rec}} = \eta \sum_t \Big(\sum_k B_{jk}^{\mathrm{random}} (y_k^{*,t} - y_k^t) \Big) \sum_{t' \leq t} \kappa^{t-t'} h_j^{t'} \hat{z}_i^{t'-1} \, . \qquad \text{(D.39)}$$

For an adaptive LIF neuron the learning rule (D.37) becomes with random feed-back weights:

$$\Delta\theta_{ji}^{\mathrm{rec}} = \eta \sum_t \Big(\sum_k B_{jk}^{\mathrm{random}} (y_k^{*,t} - y_k^t) \Big) \sum_{t' \leq t} \kappa^{t-t'} h_j^{t'} \left(\hat{z}_i^{t'-1} - \beta \epsilon_{ji,a}^{t'} \right) \, . \qquad \text{(D.40)}$$

Synaptic weight updates under *e-prop 1* for classification: For the classification tasks solved with *e-prop 1*, we consider one readout neuron y_k^t per output class, and the network output at time t corresponds to the readout with highest voltage. To train the recurrent networks in this setup, we replace the mean squared error by the the cross entropy error $E = -\sum_{t,k} \pi_k^{*,t} \log \pi_k^t$ where the target categories are provided in the form of a one-hot-encoded vector $\pi_k^{*,t}$. On the other hand, the output class distribution predicted by the network is given as $\pi_k^t = \mathrm{softmax}(y_k^t) = \exp(y_k^t) / \sum_{k'} \exp(y_{k'}^t)$. To derive the modified learning rule that results from this error function E, we replace $\frac{\partial E}{\partial z_i}$ of equation (D.32) with the corresponding gradient:

$$\frac{\partial E}{\partial z_j^t} = \theta_{kj}^{\mathrm{out}} \sum_{t' \geq t} (\pi_k^{t'} - \pi_k^{*,t'}) \kappa^{t'-t}. \tag{D.41}$$

Following otherwise the same derivation as previously it results that the weight update of *e-prop 1* previously written in equation (D.39) becomes for a LIF neuron:

$$\Delta\theta_{ji}^{\mathrm{rec}} = \eta \sum_t \big(\sum_k B_{jk}^{\mathrm{random}} (\pi_k^{*,t} - \pi_k^t) \big) \sum_{t' \leq t} \kappa^{t-t'} h_j^{t'} \hat{z}_i^{t'-1} \,. \tag{D.42}$$

Similarly, for the weight update of the output connections, the only difference between the update rules for regression and classification is that the output y_k^t and the target $y_k^{*,t}$ are respectively replaced by π_k^t and $\pi_k^{*,t}$:

$$\Delta\theta_{kj}^{\mathrm{out}} = \eta \sum_t (\pi_k^{*,t} - \pi_k^t) \sum_{t' \leq t} \kappa^{t-t'} z_j^{t'} \,. \tag{D.43}$$

Firing rate regularization: To ensure that the network computes with low firing rates, we add a regularization term E_{reg} to the error function E. This regularization term has the form:

$$E_{\mathrm{reg}} = \sum_j \left(f_j^{\mathrm{av}} - f^{\mathrm{target}} \right)^2 , \tag{D.44}$$

where f^{target} is a target firing rate and $f_j^{\mathrm{av}} = \frac{\delta t}{n_{\mathrm{trials}} T} \sum_{t,k} z_j^t$ is the firing rate of neuron j averaged over the T time steps and the n_{trials} trials separating each weight update. To compute the weight update that implements this regularization, we follow a similar derivation as detailed previously for the mean square error. Instead of equation (D.32), the partial derivative has now the form:

$$\frac{\partial E_{\mathrm{reg}}}{\partial z_j^t} = \frac{\delta t}{n_{\mathrm{trials}} T} \left(f_j^{\mathrm{av}} - f^{\mathrm{target}} \right). \tag{D.45}$$

Inserting this expression into the equation (D.33), and choosing the special case of a LIF neurons, it results that the weight update that implements the regularization is written:

$$\Delta\theta_{ji}^{\mathrm{rec}} = \eta \sum_t \frac{\delta t}{n_{\mathrm{trials}} T} \left(f^{\mathrm{target}} - f_j^{\mathrm{av}} \right) h_j^t \hat{z}_i^{t-1} \,. \tag{D.46}$$

The same learning rule is also applied to the input weights $\Delta\theta_{ji}^{\mathrm{in}}$. This weight update is performed simultaneously with the weight update exhibited in equation

(D.39) which optimizes the main error function E. For other neuron models such as adaptive LIF neurons, the equation (D.46) has to be updated accordingly to the appropriate definition of the eligibility traces.

D.2.2 Details to simulations for *E-prop 1*

General simulation and neuron parameters: In all simulations of this article, networks were simulated in discrete time with a simulation time step of 1 ms. Synapses had a uniform transmission delay of 1 ms. Synaptic weights of spiking neural networks were initialized as in (Bellec et al., 2018c).

Implementation of the optimization algorithm: A dampening factor of $\gamma = 0.3$ for the pseudo-derivative of the spiking function was used in all simulations of this article. The weights were kept constant for n_{batch} independent trials (specified for individual simulations below), and the gradients were cumulated additively. After collecting the gradients, the weights were updated using the Adam algorithm (Kingma and Ba, 2014). For all simulations of *e-prop 1*, the gradients were computed according to equation (D.35).

Integration of the "Clopath rule" in *e-prop 1*: We replaced the presynpatic and postsynaptic factors of equation (D.13) with the model of long term potentiation defined in Clopath et al., 2010 and fitted to data in the same paper. The learning rule referred as "Clopath rule" in our experiments differ from *e-prop 1* by the replacement of the pseudo derivative h_j^t by another non linear function of the post synpatic voltage. In comparison to equation (D.14) the computation of the error gradient becomes:

$$\frac{dE}{d\theta_{ji}^{\mathrm{rec}}} = \sum_t \frac{dE}{dz_j^t}[v_j^t - v_{\mathrm{th}}^+]^+[\hat{v}_j^t - v_{\mathrm{th}}^-]^+\hat{z}_i^{t-1} \ , \qquad \text{(D.47)}$$

where $\hat{v}_j^t$ is an exponential trace of the post synaptic membrane potential with time constant 10 ms and $[\cdot]^+$ is the rectified linear function. The time constant of $\hat{v}_j^t$ was chosen to match their data. The thresholds v_{th}^- and v_{th}^+ were $\frac{v_{\mathrm{th}}}{4}$ and 0 respectively. All other implementation details remained otherwise unchanged between the default implementation of *eprop 1* and this variant of the algorithm.

Pattern generation task 1.1: The three target sequences had a duration of 1000 ms and were given by the sum of four sinusoids for each sequence with fixed frequencies of 1 Hz, 2 Hz, 3 Hz, and 5 Hz. The amplitude of each sinusoidal component was drawn from a uniform distribution over the interval $[0.5, 2]$. Each component was also randomly phase-shifted with a phase sampled uniformly in the interval $[0, 2\pi)$.

The network consisted of 600 all-to-all recurrently connected LIF neurons (no adaptive thresholds). The neurons had a membrane time constant of $\tau_m = 20$ ms and a refractory period of 5 ms. The firing threshold was set to $v_{\text{th}} = 0.61$. The network outputs were provided by the membrane potential of three readout neurons with a time constant $\tau_{out} = 20$ ms. The network received input from 20 input neurons, divided into 5 groups, which indicated the current phase of the target sequence similar to (Nicola and Clopath, 2017). Neurons in group $i \in \{0, 4\}$ produced 100 Hz regular spike trains during the time interval $[200 \cdot i, 200 \cdot i + 200)$ ms and were silent at other times.

A single learning trial consisted of a simulation of the network for 1000 ms, i.e., the time to produce the target pattern at the output. The input, recurrent, and output weights of the network were trained for 1000 iterations with the Adam algorithm, a learning rate of 0.003 and the default hyperparameters (Kingma and Ba, 2014). After every 100 iterations, the learning rate was decayed with a multiplicative factor of 0.7. A batch size of a single trial was used for training. To avoid an implausibly high firing rate, a regularization term was added to the loss function, that keeps the neurons closer to a target firing rate of 10 Hz. The regularization loss was given by the mean squared error (mse) between the mean firing rate of all neurons over a batch and the target rate. This loss was multiplied with the factor 0.5 and added with the target-mse to obtain the total loss to be optimized.

The comparison algorithms in Figure 4.1d,e were implemented as follows. When training with *e-prop 1*, the random feedback weights B^{random} were generated from a Gaussian distribution with mean 0 and variance $\frac{1}{n}$, where n is the number of network neurons. For the performance of the global error signal, *e-prop 1* was used, but the random feedback matrix was replaced by a matrix where all entries had the value $\frac{1}{\sqrt{n}}$. As a second baseline a network without recurrent connections was trained with *e-prop 1* ("No rec. conn." in panel d). We further considered variants of *e-prop 1* where we sampled independent feedback matrices for every 1 or 20 ms window ("1 ms" and "20 ms" in panels d and e). Note that the same sequence of feedback matrices had to be used in every learning trials. We also compared to BPTT, where the Adam algorithm was used with the same meta-parameters as used for *e-prop 1*.

Store-recall task 1.2: The store-recall task is described in Results. Each learning trial consisted of a 2400 ms network simulation. We used a recurrent LSNN network consisting of 10 standard LIF neurons and 10 LIF neurons with adaptive thresholds. All neurons had a membrane time constant of $\tau_m = 20$ ms and a baseline threshold of $v_{\text{th}} = 0.5$. Adaptive neurons had a threshold increase constant of $\beta = 0.03$ and a threshold adaptation time constant of $\tau_a = 1200$ ms. A refractory period of 5 ms was used. The input, recurrent and output weights of the network were trained with a learning rate of 0.01 and the Adam algorithm the default hyperparameters (Kingma and Ba, 2014). Training was stopped when a misclassification rate below

0.05 was reached. After 100 iterations, the learning rate was decayed with a multiplicative factor of 0.3. The distribution of the random feedback weights B^{random} was generated from normal distribution with mean 0 and variance $\frac{1}{n}$, where n is the number of recurrent neurons. A batch size of 128 trials was used.

In Figure 4.2b, we quantified the information content of eligibility traces at training iteration 25, 75, and 200 in this task. After the predefined number of training iterations, we performed test simulations where we provided only a store command to the network and simulated the network up to 6000 ms after this store. A linear classifier (one for a time window of 100 ms, at every multiple of 50 ms) was then trained to predict the stored bit from the value of the eligibility traces at that time. For this purpose we used logistic regression with a squared regularizer on the weights. We used 150 different simulations to train the classifiers and evaluated the decoding accuracy, as shown in Figure 4.2b, on 50 separate simulations.

Speech recognition task 1.3: We followed the same task setup as in (Greff et al., 2017; Graves and Schmidhuber, 2005). The TIMIT dataset was split according to Halberstadt (Glass et al., 1999) into a training, validation, and test set with 3696, 400, and 192 sequences respectively. The networks received preprocessed audio at the input. Preprocessing of audio input consisted of the following steps: computation of 13 Mel Frequency Cepstral Coefficients (MFCCs) with frame size 10 ms on input window of 25 ms, computation of the first and the second derivatives of MFCCs, concatenation of all computed factors to 39 input channels. Input channels were mapped to the range $[0, 1]$ according to the minumum/maximum values in the training set. These continuous values were used directly as inputs x_i^t in equation (D.10).

To tackle this rather demanding benchmark task, we used a bi-directional network architecture (Graves and Schmidhuber, 2005), that is, the standard LSNN network was appended by a second network which recieved the input sequence in reverse order. A bi-directional LSNN (300 LIF neurons and 100 adaptive LIF neurons per direction) was trained with different training algorithms. Unlike in task 1.1, the random feedback weights B^{random} were generated with a variance of 1 instead of $\frac{1}{n}$ as we observed that it resulted in better performances for this task.

With LSNNs we first ran a simple 8 point grid search over the firing threshold hyperparameter v_{th}. The best performing value for threshold was then used to produce the LSNN results (see Figure 4.2c). For the strong baseline we include the result of LSTMs applied to the same task (Greff et al., 2017), where the hyperparameters were optimized using random search for 200 trials over the following hyperparameters: number of LSTM blocks per hidden layer, learning rate, momentum, momentum type, gradient clipping, and standard deviation of Gaussian input noise. In (Greff et al., 2017) the mean test accuracy of 10% best performing hyperparameter settings (out of 200) is 0.704.

Every input step which represents the 10 ms preprocessed audio frame is fed to the LSNN network for 5 consecutive 1 ms steps. All neurons had a membrane time constant of $\tau_m = 20$ ms and a refractory period of 2 ms. Adaptive neurons had $\beta = 1.8$ and an adaptation time constant of $\tau_a = 200$ ms. We used 61 readout neurons, one for each class of the TIMIT dataset. A softmax was applied to their output, which was used to compute the cross entropy error against the target label. Networks were trained using Adam with the default hyperparameters (Kingma and Ba, 2014) except $\epsilon_{\text{Adam}} = 10^{-5}$. The learning rate was fixed to 0.01 during training. We used a batch size of 32 and the membrane time constant of the output neurons was 3 ms. Regularizaion of the network firing activity was applied as in Task 1.1.

D.3 *E-prop 2*

D.3.1 Formalization and implementation of *E-prop 2*

In *e-prop 2*, the learning signals are computed in a separate error module. In order to distinguish the error module from the main network, we define a separate internal state vector for each neuron j in the error module $\boldsymbol{\sigma}_j^t$ and network dynamics $\boldsymbol{\sigma}_j^t = M_e(\boldsymbol{\sigma}_j^{t-1}, \boldsymbol{\zeta}^{t-1}, \boldsymbol{\xi}^t, \boldsymbol{\Psi})$ for it. Here, $\boldsymbol{\zeta}^{t-1}$ is the vector of neuron outputs in the error module at time t, and synaptic weights are denoted by $\boldsymbol{\Psi}$. The inputs to the error module are written as: $\boldsymbol{\xi}^t = (\boldsymbol{x}^t, \boldsymbol{z}^t, \boldsymbol{y}^{*,t})$ with $\boldsymbol{y}^{*,t}$ denoting the target signal for the network at time t. Note that the target signal is not necessarily the target output of the network, but can be more generally a target state vector of some controlled system. For example, the target signal in task 2.1 is the target position of the tip of an arm at time t, while the outputs of the network define the angular velocities of arm joints.

The error module produces at each time t a learning signal $\hat{L}_j^t$ for each neuron j of the network, which were computed according to:

$$\hat{L}_j^t = \alpha_e \hat{L}_j^{t-1} + \sum_i \Psi ji^{\text{out}} \zeta_i^t \,, \tag{D.48}$$

where the constant α_e defines the decay of the resulting learning signal, e.g. the concentration of a neuromodulator.

Synaptic weight updates under e-prop 2: The task of the error module is to compute approximations to the true learning in equation (4.4). Therefore, in comparison to equation (D.13), we obtain an estimation of the true error gradients $\frac{dE}{d\theta_{ji}^{\text{rec}}}$ given as $\widehat{\frac{dE}{d\theta_{ji}^{\text{rec}}}} = \sum_t \hat{L}_j^t h_j^t \hat{z}_i^{t-1}$, which in turn leads to an update rule for the synaptic weights using a fixed learning rate η:

$$\Delta\theta_{ji}^{\text{rec}} = -\eta \sum_t \hat{L}_j^t h_j^t \hat{z}_i^{t-1} \tag{D.49}$$

Similarly, the update rule for input weights is obtained by replacing $\hat{z}_i^{t-1}$ in favor of $\hat{x}_i^{t-1}$.

In the experiments regarding *e-prop 2*, input and recurrent weights were updated a single time in the inner loop of L2L according to $\boldsymbol{\theta}_{\text{test}} = \boldsymbol{\theta}_{\text{init}} + \Delta\boldsymbol{\theta}$, whereas output weights were kept constant.

Target movement task 2.1: In this task, the two network outputs are interpreted as angular velocities $\dot{\phi}_1$ and $\dot{\phi}_2$ and are applied to the joints of a simple arm model. The configuration of the arm model at time t is described by the angles ϕ_1^t and ϕ_2^t of the two joints measured against the horizontal and the first leg of the arm respectively, see Figure 4.3c. 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 a $\delta t = 1\,\text{ms}$. The initial values were set to $\phi_1^0 = 0$ and $\phi_2^0 = \frac{\pi}{2}$.

Feasible target movements $\boldsymbol{y}^{*,t}$ of duration 500 ms were generated randomly by sampling the generating angular velocities $\dot{\Phi}^{*,t} = (\dot{\phi}_1^{*,t}, \dot{\phi}_2^{*,t})$. Each of the target angular velocities exhibitted a common form

$$\dot{\phi}_i^{*,t} = \sum_m S_{im} \sin\left(2\pi\omega_{im}\frac{t}{T} + \delta_{im}\right) \overset{\text{def}}{=} \sum_m q_{im}^t \,, \tag{D.50}$$

where the number of components m was set to 5, S_{im} was sampled uniformly in $[0, 30]$, ω_{im} was sampled uniformly in $[0.3, 1]$ and δ_{im} was sampled uniformly in $[0, 2\pi]$. After this sampling, every component $q_{2,m}^t$ in $\dot{\phi}_2^{*,t}$ was rescaled to satisfy $\max_t(q_{2,m}^t) - \min_t(q_{2,m}^t) = 20$. In addition, we considered constraints on the angles of the joints: $\phi_1 \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and $\phi_2 \in [0, \pi]$. If violated, the respective motor commands $\dot{\phi}_i^{*,t}$ were rescaled to match the constraints.

A clock-like input signal was implemented as in task 1.1 by 20 input neurons, that fired in groups in 5 successive time steps with a length of 100 ms at a rate of 100 Hz.

Outer loop optimization: The procedure described above defines an infinitely large familiy of tasks, each task of the family being one particular target movement. We optimized the parameters of the error module as well as the initial parameters of the learning network in an outer-loop optimization procedure. The learning cost $\mathcal{L}_C$ for tasks C in the above defined family of tasks was defined as

$$\mathcal{L}_C(\boldsymbol{\theta}_{\text{test},C}) = \sum_t \left(\left(\boldsymbol{y}^t(\boldsymbol{\theta}_{\text{test},C}) - \boldsymbol{y}^{*,t}\right)^2 + \left(\dot{\Phi}^t(\boldsymbol{\theta}_{\text{test},C}) - \dot{\Phi}^{*,t}\right)^2 \right) + \lambda E_{\text{reg}} \tag{D.51}$$

to measure how well the target movement was reproduced. We then optimized the expected cost over the family of learning task using BPTT. In addition, a regularization term for firing rates as defined in equation (D.44) was introduced with $\lambda = 0.25$.

Gradients were computed over batches of 200 different tasks to empirically estimate the learning cost across the family of tasks: $\mathbb{E}_{C\sim\mathcal{F}}\left[\mathcal{L}_C(\boldsymbol{\theta}_{\text{test},C})\right] \approx \frac{1}{200}\sum_{i=1}^{200}\mathcal{L}_{C_i}(\boldsymbol{\theta}_{\text{test},C_i})$. We used the Adam algorithm (Kingma and Ba, 2014) with a learning rate of 0.0015. The learning rate decayed after every 300 steps by a factor of 0.95.

Model parameters: The learning network consisted of 400 LIF neurons according to the model stated in equation (D.10) and (D.11), with a membrane time constant of 20 ms and a threshold of $v_{\text{th}} = 0.4$. The motor commands $\dot{\phi}_j^t$ predicted by the network were given by the output of readout neurons with a membrane time constant of 20 ms. The target firing rate in the regularizer was set to $f^{target} = 20\,\text{Hz}$.

The error module was implemented as a recurrently connected network of 300 LIF neurons, which had the same membrane decay as the learning network. The neurons in the error module were set to have a threshold of $v_{\text{th}} = 0.4$. Readout neurons of the error module had a membrane time constant of 20 ms. Finally, the weight update with *e-prop* according to equation (4.7) used a learning rate of $\eta = 10^{-4}$. The target firing rate in the regularizer was set to $f^{target} = 10\,\text{Hz}$.

Both the learning network as well as the error module used a refractory period of 5 ms.

Linear error module: The alternative implementation of a linear error module was implemented as a linear mapping of inputs formerly received by the spiking implementation of the error module. Prior to the linear mapping, we applied a filter to the spiking quantities $\boldsymbol{x}^t$, $\boldsymbol{z}^t$ such that $\hat{\boldsymbol{x}}^t = \sum_{t'\leq t}\alpha_e^{t-t'}\boldsymbol{x}^t$ and similarly for $\hat{\boldsymbol{z}}^t$. Then, the learning signal from the linear error module was given as: $\hat{L}_j^t = \sum_i \Phi_{ji}^x \hat{x}_i^t + \sum_i \Phi_{ji}^z \hat{z}_i^t + \sum_i \Phi_{ji}^y y_i^{*,t}$

D.4 *E-prop 3*

D.4.1 Formalization and implementation of *E-prop 3*

We first describe *e-prop 3* in theoretical terms when the simulation duration is split into intervals of length Δt and show two mathematical properties of the algorithm: first, it computes the correct error gradients if the synthetic gradients are ideal; and second, when the synthetic gradients are imperfect, the estimated gradients are a better approximation of the true error gradient in comparison to BPTT. In subsequent paragraphs we discuss details of the implementation of *e-prop 3*, the computation of the synthetic gradients and hyperparameters used in tasks 3.1 and 3.2.

Notation and review of truncated BPTT: We consider the true error gradient $\frac{dE}{d\theta_{ji}}$ to be the error gradient computed over the full simulation ranging from time $t = 1$ to time T. Truncated BPTT computes an approximation of this gradient. In this paragraph, we identify the approximations induced by truncated BPTT.

In truncated BPTT, the network simulation is divided into K successive intervals of length Δt each. For simplicity we assume that T is a multiple of Δt, such that $K = T/\Delta t$ is an integer. Using the shorthand notation $t_m = m\Delta t$, the simulation intervals are thus $\{1, \ldots, t_1\}, \{t_1 + 1, \ldots, t_2\}, \ldots, \{t_{K-1} + 1, \ldots, t_K\}$. To simplify the theory we assume that updates are implemented after the processing of all these intervals (i.e., after time T).

For each interval $\{t_{m-1} + 1, \ldots, t_m\}$, the simulation is initialized with the network state $\boldsymbol{s}^{t_{m-1}}$. Then, the observable states $\boldsymbol{z}^{t'}$ and hidden states $\boldsymbol{s}^{t'}$ are computed for $t' \in \{t_{m-1} + 1, \ldots, t_m\}$. It is common to use for the overall error $E(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T)$ an error function that is given by the sum of errors in each individual time step. Hence the error can be written as a sum of errors $E_m(\boldsymbol{z}^{t_{m-1}+1}, \ldots, \boldsymbol{z}^{t_m})$ in the intervals:

$$E(\boldsymbol{z}^1, \ldots, \boldsymbol{z}^T) = \sum_{m=1}^{K} E_m(\boldsymbol{z}^{t_{m-1}+1}, \ldots, \boldsymbol{z}^{t_m}) \,. \tag{D.52}$$

For each such interval, after network simulation until t_m (the forward pass), the gradients $\frac{dE_m}{d\boldsymbol{s}_j^{t'}}$ are propagated backward from $t' = t_m$ to $t' = t_{m-1} + 1$ (the backward pass). The contribution to the error gradient for some paramter θ_{ji} in the interval is then given by (compare to equation (D.3))

$$g_{m,ji}^{\text{trunc}} = \sum_{t'=t_{m-1}+1}^{t_m} \frac{dE_m}{d\boldsymbol{s}_j^{t'}} \cdot \frac{\partial \boldsymbol{s}_j^{t'}}{\partial \theta_{ji}} \,. \tag{D.53}$$

The overall gradient $\frac{dE}{d\theta_{ji}}$ is then approximated by the sum of the gradients in the intervals: $g_{1,ji}^{\text{trunc}} + g_{2,ji}^{\text{trunc}} + \cdots + g_{K,ji}^{\text{trunc}}$.

This approximation is in general not equal to the true error gradient $\frac{dE}{d\theta_{ji}}$, as it disregards the contributions of network outputs within one interval on errors that occur in a later interval.

Synthetic gradients: To correct for the truncated gradient, one can provide a suitable boundary condition at the end of each interval that supplements the missing gradient. The optimal boundary condition cannot be computed in an online manner since it depends on future activities and future errors that are not yet available. In truncated BPTT, one chooses $\frac{dE}{d\boldsymbol{s}_j^{t_m+1}} = 0$ at the end of an interval $\{t_{m-1} + 1, \ldots, t_m\}$, which is exact only if the simulation terminates at time t_m or if future errors do not depend on network states of this interval. The role of synthetic gradients is to correct this approximation by providing a black box boundary condition $\text{SG}_j(\boldsymbol{z}^{t_m}, \Psi)$, where SG_j is a parameterized function of the network output with parameters Ψ. SG_j should approximate the optimal boundary condition, i.e., $\mathrm{SG}_j(\boldsymbol{z}^{t_m}, \Psi) \approx \frac{dE}{d\boldsymbol{s}_j^{t_m+1}}$.

We denote the approximate gradient that includes the boundary condition given by synthetic gradients by $\frac{d\overline{E}}{d\boldsymbol{s}_j^t}$. This gradient is given by

$$\frac{d\overline{E}_m}{d\boldsymbol{s}_j^t} = \frac{dE_m}{d\boldsymbol{s}_j^t} + \eta_{SG} \sum_l \mathrm{SG}_l(\boldsymbol{z}^{t_m}, \Psi) \frac{d\boldsymbol{s}_l^{t_m+1}}{d\boldsymbol{s}_j^t}. \tag{D.54}$$

We will continue our theoretical analysis with a factor $\eta_{SG} = 1$ (as suggested in Jaderberg et al., 2016, we set η_{SG} to 0.1 in simulations to stabilize learning). We define $g_{m,ji}^{\mathrm{SG}}$ as the corrected version of $g_{m,ji}^{\mathrm{trunc}}$ that incorporates the new boundary condition. We finally define the estimator of the error gradient with synthetic gradients as:

$$\widehat{\frac{dE}{d\theta_{ji}}}^{\mathrm{SG}} = g_{1,ji}^{\mathrm{SG}} + g_{2.ji}^{\mathrm{SG}} + \cdots + g_{K,ji}^{\mathrm{SG}}. \tag{D.55}$$

The synthetic gradient approximation is refined by minimizing the mean squared error between the synthetic gradient approximation $\mathrm{SG}_j(\boldsymbol{z}^{t_m}, \Psi)$ and the gradient $\frac{d\overline{E}_{m+1}}{d\boldsymbol{s}_j^{t_m+1}}$, which is computed in the interval $t_m + 1$ to t_{m+1} and includes the next boundary condition $\mathrm{SG}_l(\boldsymbol{z}^{t_{m+1}})$:

$$E_{\mathrm{SG}}\left(\boldsymbol{z}^{t_m}, \frac{d\overline{E}_{m+1}}{d\boldsymbol{s}_j^{t_m+1}}, \Psi\right) = \sum_j \frac{1}{2} \left\| SG_j(\boldsymbol{z}^{t_m}, \Psi) - \frac{d\overline{E}_{m+1}}{d\boldsymbol{s}_j^{t_m+1}} \right\|^2. \tag{D.56}$$

Correctness of synthetic gradients: We consider Ψ^* to be optimal synthetic gradient parameters if the synthetic gradient loss in equation (D.56) is always zero. In this case, all synthetic gradients $\mathrm{SG}(\boldsymbol{z}^{t_m}, \Psi^*)$ exactly match $\frac{dE}{d\boldsymbol{s}_j^{t_m+1}}$, and the computed approximation exactly matches the true gradient. This analysis assumes the existence of the optimal parameters Ψ^* and the convergence of the optimization algorithm to the optimal parameters. This is not necessarily true in practice. For an analysis of the convergence of the optimization of the synthetic gradient loss we refer to (Czarnecki et al., 2017).

Proof of correctness of *e-prop 3* with truncated time intervals: Similarly to the justification above for synthetic gradients, we show now that the error gradients $\frac{dE}{d\theta_{ji}}$ can be estimated with *e-prop 3* when the gradients are computed over truncated intervals.

Instead of using the factorization of the error gradients as in BPTT (equation (D.3)), *e-prop 3* uses equation (4.1). The approximate gradient that is computed by *e-prop 3* with respect to neuron outputs is given analogously to equation (D.54)

$$\frac{d\overline{E}_m}{dz_j^t} = \frac{dE_m}{dz_j^t} + \sum_l \mathrm{SG}_l(\boldsymbol{z}^{t_m}, \Psi) \frac{d\boldsymbol{s}_l^{t_m+1}}{dz_j^t}. \tag{D.57}$$

We are defining the learning signal as in equation (4.4), but now using the enhanced estimate of the derivative of the interval error:

$$\overline{\boldsymbol{L}}_{m,j}^t = \frac{d\overline{E}_m}{dz_j^t} \frac{dz_j^t}{d\boldsymbol{s}_j^t}. \tag{D.58}$$

This learning signal is computed recursively using equation (D.4) within an interval. At the upper boundaries t_m of the intervals, the boundary condition is computed via synthetic gradients.

Analogous to $g_{m,ji}^{\mathrm{SG}}$, we define the gradient approximation of *e-prop 3* $g_{m,ji}^{\mathrm{e-prop}}$ as the corrected version of $g_{m,ji}^{\mathrm{trunc}}$ that incorporates the boundary condition for interval $\{t_{m-1}+1, \dots, t_m\}$ via synthetic gradients. This gradient approximation is given by

$$g_{m,ji}^{\mathrm{e-prop}} = \sum_{t=t_{m-1}+1}^{t_m} \overline{L}_{m,j}^t \cdot \boldsymbol{\epsilon}_{ji}^t \,. \tag{D.59}$$

Considering the sum of terms $g_{m,ji}^{\mathrm{e-prop}}$ associated with each interval, we write the estimator of the true error gradient computed with *e-prop 3* as:

$$\widehat{\frac{dE}{d\theta_{ji}}}^{\mathrm{e-prop}} = g_{1,ji}^{\mathrm{e-prop}} + g_{2,ji}^{\mathrm{e-prop}} + \dots + g_{K,ji}^{\mathrm{e-prop}} \,. \tag{D.60}$$

Assuming now that this boundary condition is provided by an error module computing the synthetic gradients SG with optimal parameters Ψ^*. As explained above, it follows that all $\mathrm{SG}_l(\boldsymbol{z}^{t_m}, \Psi^*)$ computes exactly $\frac{dE}{d\boldsymbol{s}_l^{t_m+1}}$ which is true independently of the usage of BPTT or *e-prop 3*. In the later case, it follows that $\frac{d\overline{E}_m}{dz_j^t}$ is correctly computing $\frac{dE}{dz_j^t}$ and hence, $\overline{\boldsymbol{L}}_{m,j}^t$ is equal to the true learning signal $\boldsymbol{L}_j^t$. Looking back at equation (4.1), it follows that the estimator defined at equation (D.60) is equal to the true gradient if the parameters of the error module are optimal.

Optimization of the synthetic gradient parameters Ψ: We define here the algorithm used to optimize the synthetic gradients parameters Ψ and the network parameters $\boldsymbol{\theta}$. Using the same truncation scheme as described previously, we recall that the loss function $\overline{E}_m$ formalizes the loss function on interval m denoted E_m with the modification that it takes into account the boundary condition defined by the synthetic gradients. We then consider the loss E' as the sum of the term $\overline{E}_m$ and the synthetic gradient loss E_{SG}. The final algorithm is summarized by the pseudo-code given in Algorithm 4. Note that this algorithm is slightly different from the one used originally by Jaderberg et al., 2016. Our version requires one extra pair of forward and backward passes on each truncated interval but we found it easier to implement.

```
1 for $m \in \{1, \dots, K\}$ do
2     Simulate the network over the interval $\{t_{m-1}+1, \dots, t_m\}$ to compute the
        network states $\boldsymbol{s}_j^t$
3     Backpropagate gradients on the interval $\{t_{m-1}+1, \dots, t_m\}$ to compute
        $\frac{d\bar{E}_m}{d\boldsymbol{\theta}}$ using the boundary condition provided by $\mathrm{SG}_l(\boldsymbol{z}^{t_m}, \Psi)$. Store $\boldsymbol{s}_j^{t_m}$
        and $\boldsymbol{\epsilon}_{ji}^{t_m}$ to be used as initial states in the next interval.
4     Simulate the network over the interval $\{t_m+1, \dots, t_{m+1}\}$ to compute the
        network states $\boldsymbol{s}_j^t$
5     Backpropagate gradients on the interval $\{t_m+1, \dots, t_{m+1}\}$ to obtain
        $\frac{d\bar{E}_{m+1}}{d\boldsymbol{s}_j^{t_m+1}}$ and compute $\frac{dE_{\mathrm{SG}}}{d\boldsymbol{\theta}}$, $\frac{dE_{\mathrm{SG}}}{d\Psi}$,
6     Update the parameters $\Psi$ and $\boldsymbol{\theta}$ using $\frac{d(E_m+E_{\mathrm{SG}})}{d\boldsymbol{\theta}}$ and $\frac{dE_{\mathrm{SG}}}{d\Psi}$ with any
        variant of stochastic gradient descent
7 end
```

Algorithm 4: Pseudo code to describe the algorithm used to trained simultaneously the network parameters $\boldsymbol{\theta}$ and the synthetic gradients Ψ in both *e-prop 3* and BPTT with synthetic gradients.

Copy-repeat task 3.1: Each sequence of the input of the copy repeat task consists of the "8-bit" pattern of length n_{pattern} encoded by 8 binary inputs, a stop character encoded by a 9th binary input channel, and a number of repetitions $n_{\mathrm{repetitions}}$ encoded using a one hot encoding over the 9 input channels. While the input is provided, no output target is defined. After that the input becomes silent and the output target is defined by the $n_{\mathrm{repetitions}}$ copies of the input followed by a stop character. As for the input, the output pattern is encoded with the first 8 output channels and the 9-th channel is used for the stop character. Denoting the target output $b_k^{*,t}$ of the channel k at time t and defining $\sigma(y_k^t)$ as the output of the network with y_k^t a weighted sum of the observable states z_j^t and σ the sigmoid function, the loss function is defined by the binary cross-entropy loss: $E = -\sum_{t,k}(1-b_k^{*,t})\log_2 \sigma(y_k^t) + b_k^{*,t}\log_2\left(1-\sigma(y_j^t)\right)$. The sum is running over the time steps where the output is specified.

We follow the curriculum of Jaderberg et al., 2016 to increase gradually the complexity of the task: when the error E averaged over a batch of 256 sequences is below 0.15 bits per sequences, n_{pattern} or $n_{\mathrm{repetitions}}$ are incremented by one. When the experiments begins, we initialize n_{pattern} and $n_{\mathrm{repetitions}}$ to one. After the first threshold crossing n_{pattern} is incremented, then the increments are alternating between n_{pattern} and $n_{\mathrm{repetitions}}$.

For each batch of 256 sequences, the parameters are updated every $\Delta t = 4$ time steps when the simulation duration in truncated as in BPTT. The parameter updates are applied with Adam, using learning rate 0.0001 and the default hyperparameters suggested by Kingma and Ba, 2014.

Word prediction task 3.2: Training was performed for 20 epochs, where one epoch denotes a single pass through the complete dataset. All learning rules used gradient descent to minimize loss with initial learning rate of 1 which was decayed after every epoch with factor 0.5, starting with epoch 5. Mini-batch consisted of 20 sequences of length Δt. Sequence of sentences in Penn Treebank dataset are connected and coherent, so the network state was reset only after every epoch. Equally the eligibility traces are set to zero at the beginning of every epoch.

Acknowledgments

This research was supported by the Human Brain Project of the European Union, Grant agreement No. 785907. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro P6000 GPU used for this research. Computations were primarily carried out on the Supercomputer JUWELS at Jülich Supercomputing Centre. We gratefully acknowledge the support of the SimLab of the Forschungszentrum Jülich in securing the grant CHHD34 "Learning to Learn on Spiking Neural Networks" from the Gauss Centre for Supercomputing, which provided funding for this computing time. We also gratefully acknowledge the Vienna Scientific Cluster (VSC) for providing additional computing time. We would like to thank Arjun Rao for his contribution to the software used in our experiments. We also want to thank him, Mike Davies, Michael Müller, Christoph Stoeckl, and Anand Subramoney for comments on earlier version of the manuscript.

Bibliography

Allen Institute: Cell Types Database (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. 5, 41).

AllenInstitute (2017). "Allen Cell Types Database, technical white paper: Electrophysiology." In: (cit. on pp. 4, 5).

Attardo, A., J. E. Fitzgerald, and M. J. Schnitzer (2015). "Impermanence of dendritic spines in live adult CA1 hippocampus." In: *Nature* 523.7562, pp. 592–596 (cit. on p. 14).

Barrett, D. G., F. Hill, A. Santoro, A. S. Morcos, and T. Lillicrap (2018). "Measuring abstract reasoning in neural networks." In: *arXiv preprint arXiv:1807.04225* (cit. on p. 70).

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. 30, 33, 68, 95, 97, 102).

Bellec, G., D. Salaj, A. Subramoney, R. Legenstein, and W. Maass (2018b). "Computational properties of networks of spiking neurons with adapting neurons; in preparation." In: (cit. on p. 31).

Bellec, G., D. Salaj, A. Subramoney, R. Legenstein, and W. Maass (2018c). "Long short-term memory and learning-to-learn in networks of spiking neurons." In: *NeurIPS 32* (cit. on pp. 48, 50, 53, 57, 68, 107, 114).

Bellec, G., F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass (2019). "A solution to the learning dilemma for recurrent networks of spiking neurons." In: *bioRxiv*, p. 738385 (cit. on pp. 6–10).

Bengio, Y., N. Léonard, and A. Courville (2013). "Estimating or propagating gradients through stochastic neurons for conditional computation." In: *arXiv preprint arXiv:1308.3432* (cit. on p. 6).

Bengio, Y., P. Simard, and P. Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult." In: *Neural Networks, IEEE Transactions on* 5.2, pp. 157–166 (cit. on pp. 7, 95).

Bi, G. Q. and M. M. Poo (1998). "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type." In: *Journal of Neuroscience* 18, pp. 10464–10472 (cit. on p. 7).

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer (cit. on p. 15).

Bliss, T. V. P. and T. Lomo (1973). "Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the preforant path." In: *Journal of physiology* 232, pp. 331–356 (cit. on p. 7).

Brea, J. and W. Gerstner (2016). "Does computational neuroscience need new synaptic learning paradigms?" In: *Current Opinion in Behavioral Sciences* 11, pp. 61–66 (cit. on pp. 58, 69).

Brette, R. and W. Gerstner (2005). “Adaptive exponential integrate-and-fire model as an effective description of neuronal activity.” In: *Journal of neurophysiology* 94.5, pp. 3637–3642 (cit. on p. 5).

Buzsaki, G. (2006). *Rhythms of the Brain*. Oxford University Press (cit. on p. 44).

Buzzell, G. A., J. E. Richards, L. K. White, T. V. Barker, D. S. Pine, and N. A. Fox (2017). “Development of the error-monitoring system from ages 9–35: Unique insight provided by MRI-constrained source localization of EEG.” In: *Neuroimage* 157, pp. 13–26 (cit. on p. 45).

Chambers, A. R. and S. Rumpel (2017). “A stable brain from unstable components: Emerging concepts, implications for neural computation.” In: *Neuroscience* (cit. on p. 14).

Chen, C., D. Carlson, Z. Gan, C. Li, and L. Carin (2016). “Bridging the gap between stochastic gradient MCMC and stochastic optimization.” In: *Artificial Intelligence and Statistics*, pp. 1051–1060 (cit. on pp. 16, 26).

Chen, C. H., R. Fremont, E. E. Arteaga-Bracho, and K. Khodakhah (2014). “Short latency cerebellar modulation of the basal ganglia.” In: *Nature Neuroscience* 17.12, p. 1767 (cit. on p. 69).

Clopath, C., L. Büsing, E. Vasilaki, and W. Gerstner (2010). “Connectivity reflects coding: a model of voltage-based STDP with homeostasis.” In: *Nature Neuroscience* 13.3, pp. 344–52 (cit. on pp. 9, 47, 49, 52, 54, 69, 114).

Collins, M. D. and P. Kohli (2014). “Memory bounded deep convolutional networks.” In: *arXiv preprint arXiv:1412.1442* (cit. on pp. 14, 21, 22, 76).

Costa, R., I. A. Assael, B. Shillingford, N. de Freitas, and T. Vogels (2017). “Cortical microcircuits as gated-recurrent neural networks.” In: *Advances in Neural Information Processing Systems*, pp. 272–283 (cit. on p. 33).

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 pp. 30, 32, 94).

Cui, Y., I. Prokin, H. Xu, B. Delord, S. Genet, L. Venance, and H. Berry (2016). “Endocannabinoid dynamics gate spike-timing dependent depression and potentiation.” In: *Elife* 5, e13185 (cit. on p. 69).

Czarnecki, W. M., G. Świrszcz, M. Jaderberg, S. Osindero, O. Vinyals, and K. Kavukcuoglu (2017). “Understanding synthetic gradients and decoupled neural interfaces.” In: *arXiv preprint arXiv:1703.00522* (cit. on pp. 45, 47, 48, 63, 69, 121).

D’Angelo, E., L. Mapelli, C. Casellato, J. A. Garrido, N. Luque, J. Monaco, F. Prestori, A. Pedrocchi, and E. Ros (2016). “Distributed circuit plasticity: new clues for the cerebellar mechanisms of learning.” In: *The Cerebellum* 15.2, pp. 139–151 (cit. on p. 46).

Davies, M., N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, et al. (2018). “Loihi: A neuromorphic manycore processor with on-chip learning.” In: *IEEE Micro* 38.1, pp. 82–99 (cit. on pp. 11, 31, 47).

DePasquale, B., M. M. Churchland, and L. Abbott (2016). "Using firing-rate dynamics to train recurrent networks of spiking model neurons." In: *arXiv preprint arXiv:1601.07620* (cit. on p. 31).

Duan, Y., J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel (2016). "RL2: Fast Reinforcement Learning via Slow Reinforcement Learning." In: *arXiv preprint arXiv:1611.02779* (cit. on pp. 31, 35, 36, 39, 46).

Eliasmith, C. (2013). *How to build a brain: A neural architecture for biological cognition*. Oxford University Press (cit. on p. 31).

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 pp. 8, 10, 46, 58).

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 (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 pp. 6, 30–32, 94).

Foncelle, A., A. Mendes, J. Jedrzejewska-Szmek, S. Valtcheva, H. Berry, K. Blackwell, and L. Venance (2018). "Modulation of spike-timing dependent plasticity: towards the inclusion of a third factor in computational models." In: *Frontiers in computational neuroscience* 12, p. 49 (cit. on p. 69).

Freitas, J. F. de, M. Niranjan, A. H. Gee, and A. Doucet (2000). "Sequential Monte Carlo methods to train neural network models." In: *Neural computation* 12.4, pp. 955–993 (cit. on pp. 15, 26).

Frémaux, N. and W. Gerstner (2016). "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules." In: *Frontiers in neural circuits* 9, p. 85 (cit. on pp. 7, 47, 52).

Frémaux, N., H. Sprekeler, and W. Gerstner (2013). "Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons." In: *{PLoS} Comput Biol* 9.4, e1003024 (cit. on p. 8).

Furber, S. B., F. Galluppi, S. Temple, and L. A. Plana (2014). "The spinnaker project." In: *Proceedings of the IEEE* 102.5, pp. 652–665 (cit. on pp. 14, 47).

Furber, S. B., D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown (2013). "Overview of the spinnaker system architecture." In: *IEEE Transactions on Computers* 62.12, pp. 2454–2467 (cit. on p. 31).

Gardiner, C. W. et al. *Handbook of stochastic methods*. Vol. 3 (cit. on pp. 81, 82).

Gehring, W. J., B. Goss, M. G. Coles, D. E. Meyer, and E. Donchin (1993). "A neural system for error detection and compensation." In: *Psychological science* 4.6, pp. 385–390 (cit. on p. 45).

Gerstner, W., R. Kempter, J. L. van Hemmen, and H. Wagner (1996). "A neuronal learning rule for sub-millisecond temporal coding." In: *Nature* 383.6595, p. 76 (cit. on p. 7).

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 pp. 4, 32).

Gerstner, W., M. Lehmann, V. Liakoni, D. Corneil, and J. Brea (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. 7, 8, 10, 46, 47, 52, 53).

Glass, J., A. Smith, and A. K. Halberstadt (1999). "Heterogeneous Acoustic Measurements and Multiple Classifiers for Speech Recognition." In: (cit. on pp. 97, 116).

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 p. 100).

Gouwens, N. W., J. Berg, D. Feng, S. A. Sorensen, H. Zeng, M. J. Hawrylycz, C. Koch, and A. Arkhipov (2018). "Systematic generation of biophysically detailed models for diverse cortical neuron types." In: *Nature communications* 9.1, p. 710 (cit. on p. 32).

Graves, A., A.-R. Mohamed, and G. Hinton (2013). "Speech recognition with deep recurrent neural networks." In: *ICASSP*, pp. 6645–6649 (cit. on pp. 20, 77).

Graves, A. and J. Schmidhuber (2005). "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." In: *Neural Networks* 18.5-6, pp. 602–610 (cit. on pp. 20, 116).

Graves, A., G. Wayne, and I. Danihelka (2014). "Neural turing machines." In: *arXiv preprint arXiv:1410.5401* (cit. on pp. 62, 65).

Greff, K., R. K. Srivastava, J. Koutnık, B. R. Steunebrink, and J. Schmidhuber (2017). "LSTM: A search space odyssey." In: *IEEE TNNLS* 28.10, pp. 2222–2232 (cit. on pp. 20, 35, 58, 77, 78, 116).

Gu, S., S. Levine, I. Sutskever, and A. Mnih (2015). "MuProp: Unbiased backpropagation for stochastic neural networks." In: *arXiv preprint arXiv:1511.05176* (cit. on p. 6).

Han, S., H. Mao, and W. J. Dally (2015a). "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." In: *arXiv preprint arXiv:1510.00149* (cit. on p. 14).

Han, S., J. Pool, J. Tran, and W. Dally (2015b). "Learning both weights and connections for efficient neural network." In: *Advances in Neural Information Processing Systems*, pp. 1135–1143 (cit. on pp. 14, 19, 21, 22, 76, 77).

Han, S., J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, et al. (2017). "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA." In: *FPGA*, pp. 75–84 (cit. on p. 14).

Hasson, U., J. Chen, and C. J. Honey (2015). "Hierarchical process memory: memory as an integral component of information processing." In: *Trends in cognitive sciences* 19.6, pp. 304–313 (cit. on p. 30).

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory." In: *Neural Computation* 9.8, pp. 1735–1780 (cit. on pp. 2, 48, 109).

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 p. 36).

Hodgkin, A. L. (1958). "The Croonian Lecture-Ionic movements and electrical activity in giant nerve fibres." In: *Proceedings of the Royal Society of London. Series B-Biological Sciences* 148.930, pp. 1–37 (cit. on pp. 3, 4).

Holtmaat, A. J., J. T. Trachtenberg, L. Wilbrecht, G. M. Shepherd, X. Zhang, G. W. Knott, and K. Svoboda (2005). "Transient and persistent dendritic spines in the neocortex in vivo." In: *Neuron* 45.2, pp. 279–291 (cit. on p. 14).

Hosp, J. A., M. S. Pekanovic A. Rioult-Pedotti, and A. R. Luft (2011). "Dopaminergic Projections from Midbrain to Primary Motor Cortex Mediate Motor Skill Learning." In: *The Journal of Neuroscience* 31.7, pp. 2481–24887 (cit. on p. 46).

Huh, D. and T. J. Sejnowski (2017). "Gradient descent for spiking neural networks." In: *arXiv preprint arXiv:1706.04698* (cit. on pp. 7, 31, 32).

Iandola, F. N., S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer (2016). "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size." In: *arXiv preprint arXiv:1602.07360* (cit. on p. 14).

Izhikevich, E. M. (2007). "Solving the distal reward problem through linkage of STDP and dopamine signaling." In: *Cerebral cortex* 17.10, pp. 2443–2452 (cit. on p. 8).

Jaderberg, M., W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu (2016). "Decoupled neural interfaces using synthetic gradients." In: *arXiv preprint arXiv:1608.05343* (cit. on pp. 47, 48, 62–66, 69, 121, 123).

Jin, X., X. Yuan, J. Feng, and S. Yan (2016). "Training Skinny Deep Neural Networks with Iterative Hard Thresholding Methods." In: *arXiv preprint arXiv:1607.05423* (cit. on p. 14).

Jouppi, N. P., C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. (2017). "In-datacenter performance analysis of a tensor processing unit." In: *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, pp. 1–12 (cit. on p. 14).

Kaiser, J., H. Mostafa, and E. Neftci (2018). "Synaptic Plasticity Dynamics for Deep Continuous Local Learning." In: *arXiv preprint arXiv:1811.10766* (cit. on p. 53).

Kandel, E. R., J. H. Schwartz, T. M. Jessell, D. of Biochemistry, M. B. T. Jessell, S. Siegelbaum, and A. Hudspeth (2000). *Principles of neural science*. Vol. 4. McGraw-hill New York (cit. on p. 44).

Kappel, D., S. Habenschuss, R. Legenstein, and W. Maass (2015). "Network Plasticity as Bayesian Inference." In: *PLOS Computational Biology* 11.11, e1004485 (cit. on pp. 16, 24, 33, 80, 82).

Kappel, D., R. Legenstein, S. Habenschuss, M. Hsieh, and W. Maass (2018). "Reward-based stochastic self-configuration of neural circuits." In: *eNEURO* (cit. on pp. 8, 30, 32, 33, 80).

Kingma, D. P. and J. Ba (2014). "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (cit. on pp. 15, 78, 99, 100, 114, 115, 117, 119, 124).

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*, pp. 1097–1105 (cit. on p. 6).

Lake, B. M., T. D. Ullman, T. J. B., and S. J. Gershman (2017). "Building machines that learn and think like peole." In: *Behavioral and Brain Sciences* 40 (cit. on p. 69).

Le, Q. V., N. Jaitly, and G. E. Hinton (2015). "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units." In: *CoRR* abs/1504.00941 (cit. on p. 33).

LeCun, Y., Y. Bengio, and G. Hinton (2015). "Deep learning." In: *nature* 521.7553, p. 436 (cit. on pp. 1, 2, 6, 44).

Lee, K.-F. and H.-W. Hon (1989). "Speaker-independent phone recognition using hidden Markov models." In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.11, pp. 1641–1648 (cit. on p. 77).

Legenstein, R. and W. Maass (2007). "Edge of chaos and prediction of computational performance for neural circuit models." In: *Neural Networks* 20.3, pp. 323–334 (cit. on p. 6).

Legenstein, R., D. Pecevski, and W. Maass (2008). "A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback." In: *PLoS computational biology* 4.10, e1000180 (cit. on p. 8).

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 pp. 8, 53).

Liu, C., G. Bellec, B. Vogginger, D. Kappel, J. Partzsch, F. Neumärker, S. Höppner, W. Maass, S. B. Furber, R. Legenstein, et al. (2018). "Memory-efficient deep learning on a SpiNNaker 2 prototype." In: *Frontiers in neuroscience* 12 (cit. on p. 11).

Lorente de Nó, R. (1938). "Architectonics and structure of the cerebral cortex." In: *Physiology of the nervous system*, pp. 291–330 (cit. on p. 44).

Maass, W., T. Natschläger, and H. Markram (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 p. 5).

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. 45).

Markram, H., E. Muller, S. Ramaswamy, M. W. Reimann, M. Abdellah, C. A. Sanchez, A. Ailamaki, L. Alonso-Nanclares, N. Antille, S. Arsever, et al. (2015). "Reconstruction and simulation of neocortical microcircuitry." In: *Cell* 163.2, pp. 456–492 (cit. on p. 4).

Merolla, P. A., J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, et al. (2014). "A million spiking-neuron integrated circuit with a scalable communication network and interface." In: *Science* 345.6197, pp. 668–673 (cit. on p. 14).

Mikolov, T., A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato (2014). "Learning longer memory in recurrent neural networks." In: *arXiv preprint arXiv:1412.7753* (cit. on p. 32).

Mongillo, G., O. Barak, and M. Tsodyks (2008). "Synaptic theory of working memory." eng. In: *Science (New York, N.Y.)* 319.5869, pp. 1543–1546 (cit. on pp. 6, 42).

Mongillo, G., S. Rumpel, and Y. Loewenstein (2017). "Intrinsic volatility of synaptic connections - a challenge to the synaptic trace theory of memory." In: *Current Opinion in Neurobiology* 46, pp. 7–13 (cit. on p. 26).

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 p. 39).

Narang, S., G. Diamos, S. Sengupta, and E. Elsen (2017). "Exploring Sparsity in Recurrent Neural Networks." In: *arXiv preprint arXiv:1704.05119* (cit. on p. 14).

Neal, R. M. (1992). *Bayesian training of backpropagation networks by the hybrid Monte Carlo method*. Tech. rep. Technical Report CRG-TR-92-1, Dept. of Computer Science, University of Toronto (cit. on pp. 15, 83).

Nevian, T. and B. Sakmann (2006). "Spine Ca2+ signaling in spike-timing-dependent plasticity." In: *Journal of Neuroscience* 26.43, pp. 11001–11013 (cit. on p. 47).

Ngezahayo, A., M. Schachner, and A. Artola (2000). "Synaptic activity modulates the induction of bidirectional synaptic changes in adult mouse hippocampus." In: *Journal of Neuroscience* 20.7, pp. 2451–2458 (cit. on p. 47).

Nicola, W. and C. Clopath (2017). "Supervised learning in spiking neural networks with FORCE training." In: *Nature Communications* 8.1, p. 2208 (cit. on pp. 7, 31, 53, 57, 68, 115).

Nøkland, A. (2016). "Direct feedback alignment provides learning in deep neural networks." In: *NIPS*, pp. 1037–1045 (cit. on pp. 46, 52, 68, 112).

Paille, V., E. Fino, K. Du, T. Morera-Herreras, S. Perez, J. H. Kotaleski, and L. Venance (2013). "GABAergic circuits control spike-timing-dependent plasticity." In: *Journal of Neuroscience* 33.22, pp. 9353–9363 (cit. on p. 69).

Patterson, S. and Y. W. Teh (2013). "Stochastic gradient Riemannian Langevin dynamics on the probability simplex." In: *Advances in Neural Information Processing Systems*, pp. 3102–3110 (cit. on p. 26).

Perich, M. G., J. A. Gallego, and L. E. Miller (2018). "A neural population mechanism for rapid learning." In: *Neuron* (cit. on p. 36).

Perrin, E. and L. Venance (2019). "Bridging the gap between striatal plasticity and learning." In: *Current opinion in neurobiology* 54, pp. 104–112 (cit. on p. 69).

Pi, H., B. Hangya, D. Kvitsiani, J. I. Sanders, Z. J. Huang, and A. Kepecs (2013). "Cortical interneurons that specialize in disinhibitory control." In: *Nature* 503.7477, pp. 521–524 (cit. on p. 46).

Pillow, J. W., J. Shlens, L. Paninski, A. Sher, A. M. Litke, E. Chichilnisky, and E. P. Simoncelli (2008). "Spatio-temporal correlations and visual signalling in a complete neuronal population." In: *Nature* 454.7207, p. 995 (cit. on pp. 4, 5).

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, e1004275 (cit. on pp. 5, 32).

Pozzorini, C., R. Naud, S. Mensi, and W. Gerstner (2013). "Temporal whitening by power-law adaptation in neocortical neurons." In: *Nature neuroscience* 16.7, pp. 942–8 (cit. on p. 5).

Purves, D., G. J. Augustine, D. Fitzpatrick, W. C. Hall, A.-S. LaMantia, J. O. McNamara, and L. E. White (2008). "Neuroscience. 4th." In: *Sunderland, Mass.: Sinauer. xvii* 857, p. 944 (cit. on pp. 3, 4).

Qiao, N., H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri (2015). "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses." In: *Frontiers in neuroscience* 9, p. 141 (cit. on p. 31).

Raiko, T., M. Berglund, G. Alain, and L. Dinh (2014). "Techniques for learning binary stochastic feedforward neural networks." In: *arXiv preprint arXiv:1406.2989* (cit. on p. 6).

Rajan, K. and L. F. Abbott (2006). "Eigenvalue spectra of random matrices for neural networks." In: *Physical review letters* 97.18, p. 188104 (cit. on p. 95).

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. 100).

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science (cit. on p. 83).

Sajad, A., D. C. Godlove, and J. D. Schall (2019). "Cortical microcircuitry of performance monitoring." En. In: *Nature Neuroscience* 22.2, p. 265 (cit. on p. 10).

Samadi, A., T. P. Lillicrap, and D. B. Tweed (2017). "Deep learning with dynamic spiking neurons and fixed feedback weights." In: *Neural computation* 29.3, pp. 578–602 (cit. on pp. 46, 52, 68, 112).

Scellier, B. and Y. Bengio (2017). "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation." In: *Frontiers in computational neuroscience* 11, p. 24 (cit. on p. 8).

Scellier, B. and Y. Bengio (2019). "Equivalence of equilibrium propagation and recurrent backpropagation." In: *Neural computation* 31.2, pp. 312–329 (cit. on p. 8).

Schemmel, J., D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner (2010). "A wafer-scale neuromorphic hardware system for large-scale neural modeling." In: *Circuits and systems (ISCAS), proceedings of 2010 IEEE international symposium on*. IEEE, pp. 1947–1950 (cit. on pp. 14, 31, 47).

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. 40, 101).

Schultz, W. (2002). "Getting formal with dopamine and reward." In: *Neuron* 36.2, pp. 241–263 (cit. on pp. 7, 8).

Schultz, W., P. Dayan, and P. R. Montague (1997). "A neural substrate of prediction and reward." In: *Science* 275.5306, pp. 1593–1599 (cit. on p. 8).

Seung, H. S. and U. Sümbül (2014). "Neuronal cell types and connectivity: lessons from the retina." In: *Neuron* 83.6, pp. 1262–1272 (cit. on p. 4).

Sjöström, P. J., G. G. Turrigiano, and S. B. Nelson (2001). "Rate, timing, and cooperativity jointly determine cortical synaptic plasticity." In: *Neuron* 32.6, pp. 1149–1164 (cit. on p. 47).

Srinivas, S. and R. V. Babu (2015). "Data-free parameter pruning for deep neural networks." In: *arXiv preprint arXiv:1507.06149* (cit. on p. 14).

Stettler, D. D., H. Yamahachi, W. Li, W. Denk, and C. D. Gilbert (2006). "Axons and synaptic boutons are highly dynamic in adult visual cortex." In: *Neuron* 49.6, pp. 877–887 (cit. on p. 14).

Stokes, M. G. (2015). "'Activity-silent'working memory in prefrontal cortex: a dynamic coding framework." In: *Trends in Cognitive Sciences* 19.7, pp. 394–405 (cit. on p. 42).

Subramoney, A., G. Bellec, F. Scherr, R. Legenstein, and W. Maass (2018). "Recurrent networks of spiking neurons learn to learn; in preparation." In: (cit. on pp. 35, 42).

Sugihara, H., N. Chen, and M. Sur (2016). "Cell-specific modulation of plasticity and cortical state by cholinergic inputs to the visual cortex." In: *Journal of Physiology* 110.1-2, pp. 37–43 (cit. on p. 46).

Sussillo, D. and L. F. Abbott (2009). "Generating coherent patterns of activity from chaotic neural networks." In: *Neuron* 63.4, pp. 544–557 (cit. on p. 7).

Sussillo, D. and L. Abbott (2014). "Random walk initialization for training very deep feedforward networks." In: *arXiv preprint arXiv:1412.6558* (cit. on p. 95).

Sussillo, D., T. Toyoizumi, and W. Maass (2007). "Self-Tuning of Neural Circuits Through Short-Term Synaptic Plasticity." en. In: *Journal of Neurophysiology* 97.6, pp. 4079–4095 (cit. on p. 6).

Sutton, R. S. and A. G. Barto (1998). *Introduction to reinforcement learning*. Vol. 135. MIT Press Cambridge (cit. on p. 47).

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* 1.1, pp. 1–15 (cit. on p. 32).

Tibshirani, R. (1996). "Regression shrinkage and selection via the lasso." In: *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288 (cit. on pp. 21, 22, 76).

Tieleman, T. and G. Hinton (2012). "Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude." In: *COURSERA: Neural networks for machine learning* 4.2, pp. 26–31 (cit. on p. 15).

Trachtenberg, J. T., B. E. Chen, G. W. Knott, G. Feng, J. R. Sanes, E. Welker, and K. Svoboda (2002). "Long-term in vivo imaging of experience-dependent synaptic plasticity in adult cortex." In: *Nature* 420.6917, p. 788 (cit. on p. 4).

Tsodyks, M., K. Pawelzik, and H. Markram (1998). "Neural networks with dynamic synapses." In: *Neural computation* 10.4, pp. 821–835 (cit. on p. 5).

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 p. 39).

Wang, J. X., Z. Kurth-Nelson, D. Kumaran, D. Tirumala, H. Soyer, J. Z. Leibo, D. Hassabis, and M. Botvinick (2018). "Prefrontal cortex as a meta-reinforcement learning system." In: *Nature Neuroscience* 21.6, p. 860 (cit. on pp. 31, 35, 36, 39).

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. 31, 35, 36, 39, 46).

Wang, Z., S. Joshi, S. Savel'ev, W. Song, R. Midya, Y. Li, M. Rao, P. Yan, S. Asapu, Y. Zhuo, et al. (2018). "Fully memristive neural networks for pattern classification with unsupervised learning." In: *Nature Electronics* 1.2, p. 137 (cit. on p. 70).

Welling, M. and Y. W. Teh (2011). "Bayesian learning via stochastic gradient Langevin dynamics." In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 681–688 (cit. on pp. 15, 16, 26).

Werbos, P. J. (1990). "Backpropagation through time: what it does and how to do it." In: *Proceedings of the IEEE* 78.10, pp. 1550–1560 (cit. on pp. 50, 105).

Williams, R. J. and D. Zipser (1989). "A learning algorithm for continually running fully recurrent neural networks." In: *Neural Computation* 1.2, pp. 270–280 (cit. on pp. 50, 51).

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 pp. 7, 8).

Yamins, D. L. and J. J. DiCarlo (2016). "Using goal-driven deep learning models to understand sensory cortex." In: *Nature neuroscience* 19.3, p. 356 (cit. on p. 4).

Yang, Y., M. Yin, Z. Yu, Z. Wang, T. Zhang, Y. Cai, W. D. Lu, and R. Huang (2017). "Multifunctional Nanoionic Devices Enabling Simultaneous Heterosynaptic Plasticity and Efficient In-Memory Boolean Logic." In: *Advanced Electronic Materials* 3.7, p. 1700032 (cit. on p. 70).

Yang, Z., M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang (2015). "Deep fried convnets." In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1476–1483 (cit. on p. 14).

Yu, D., F. Seide, G. Li, and L. Deng (2012). "Exploiting sparseness in deep neural networks for large vocabulary speech recognition." In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4409–4412 (cit. on p. 22).

Zenke, F. and S. Ganguli (2018). "Superspike: Supervised learning in multilayer spiking neural networks." In: *Neural computation* 30.6, pp. 1514–1541 (cit. on pp. 8, 53).